

PROCEEDINGS OF THE

*10<sup>th</sup>* INTERNATIONAL  
**MODELICA**  
CONFERENCE

March 10-12, 2014  
Lund, Sweden  
[www.modelica.org](http://www.modelica.org)



EDITORS: HUBERTUS TUMMESCHEIT AND KARL-ERIK ÅRZÉN

*Modelon*



*m*  
MODELICA

The Conference is organized by Modelon in collaboration with the Linnaeus center LCC at Lund University in cooperation with the Modelica Association.

**Proceedings of the 10<sup>th</sup> International Modelica Conference**  
Lund, Sweden, March 10-12, 2014

**Editors:**

Dr. Hubertus Tummescheit and Prof. Karl-Erik Årzén

**Published by:**

Modelica Association and Linköping University Electronic Press

ISBN: 978-91-7519-380-9

Series: Linköping Electronic Conference Proceedings, No 96

ISSN: 1650-3686

eISSN: 1650-3740

DOI: <http://dx.doi.org/10.3384/ecp14096>

**Organized by:**

Modelica Association  
c/o PELAB, Linköpings Univ.  
SE-581 83 Linköping  
Sweden

and Modelon AB  
IDEON Science Park  
SE-223 70 Lund  
Sweden

**Conference location:**

Lund University  
Matematikcentrum/Matteannexet  
(Center for Mathematical Sciences)  
Sölvegatan 20A, SE-223 62 LUND  
SWEDEN

Copyright © Modelica Association, 2014

## Welcome

The 10<sup>th</sup> International Modelica Conference is the main event for our community. Users, library developers, tool vendors, and language designers will gather to share their knowledge and learn about the latest scientific and industrial progress related to Modelica and FMI (Functional Mockup Interface).

This 10<sup>th</sup> milestone conference returns to Lund, where the first event took place in 2000. Since then, Modelica has matured from an idea among a small number of dedicated enthusiasts to a widely accepted standard language for the modeling and simulation of cyber-physical systems. Modelica is now used in many industries including automotive, energy and process, aerospace, and industrial equipment. Modelica has even been tapped for one-of-a-kind systems engineering designs such as the ESS (European Spallation Source) which is currently being built nearby in Lund. Modelica is the language of choice for modeling and simulation of complex system interactions.

The addition of the FMI standard to the project portfolio under the stewardship of the Modelica Association has greatly strengthened Modelica. FMI provides a complementary standard that enables deployment of high quality models to a larger number of engineers working with system design and verification.

Conference highlights:

- 2 Keynote speeches
- 114 papers in 5 parallel tracks
- 23 posters
- 6 tutorials
- 5 libraries for the Modelica Library Award
- 6 vendor sessions presenting the latest Modelica and FMI tools
- A fully booked exhibition area featuring 18 exhibitors
- Electronic proceedings including all papers and some associated Modelica libraries and models

The conference also presents new initiatives from the Modelica Association. Since the last conference, there has been a major effort to improve the standards compliance process for the Modelica language, the Modelica Libraries developed by the Modelica association, and the FMI standard.

- The latest Modelica Standard Library release (MSL 3.2.1) has been enhanced and modified to be fully compliant with the Modelica Language Standards version 3.2 rev2, and is now solely based on open source code under the Modelica License version 2.0.
- MSL 3.2.1 has also been improved to significantly simplify comparisons of simulations of the same model across multiple Modelica environments. Tools to support such comparisons are now available through the Modelica Association.
- The Modelica language version 3.2rev 2 fixed many ambiguities in the specification.
- A Modelica Compliance Test Library has been carefully designed and implemented to verify that a Modelica tool is compliant to the Modelica specification. It has been tested with many tools, with agreed-upon reference results.
- A set of FMI Cross Check Rules was established in July 2013 and has been used by many vendors to verify tool quality and interoperability. All results are publically presented in a dynamic, online, tabular reference.

These combined efforts have helped to increase the industrial acceptance, commitment to, and use of Modelica and FMI as central standards for analytic model based systems engineering.

Finally, we want to acknowledge the support we received from the program board and program committee. Special thanks to this year's organizers, the Modelica Association, Modelon AB, and Amelie Rönngård from Anagram. Last but not least, let us thank all authors for their contributions to this conference.

We wish all participants an enjoyable and successful conference.

West Hartford and Lund, February 10<sup>th</sup> 2014

Hubertus Tummescheit and Karl-Erik Årzén

## Program Committee

### Program Chairs

Dr. Hubertus Tummescheit, Modelon Inc., USA  
Prof. Karl-Erik Årzén, Lund University, Sweden

### Program Board

Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden  
Prof. Peter Fritzson, Linköping University, Sweden  
Prof. Martin Otter, DLR, Germany  
Dr. Michael Tiller, Xogeny, Michigan, USA

### Program Committee

Dr. Johan Åkesson, Modelon AB, Lund, Sweden  
Prof. Karl-Erik Årzén, Lund University, Lund, Sweden  
Prof. Bernhard Bachmann, Univ. Applied Sciences Bielefeld, Bielefeld, Germany  
Dr. John Batteh, Modelon Inc., Ann Arbor, USA  
Dr. Albert Benveniste, INRIA, Rennes, France  
Christian Bertsch, Robert Bosch GmbH, Stuttgart, Germany  
Volker Beuter, VI-grade GmbH, Marburg, Germany  
Torsten Blochwitz, ITI GmbH, Dresden, Germany  
Dr. Scott Bortoff, MERL Cambridge, USA  
Daniel Bouskela, EDF R&D, Paris, France  
Dr. David Broman, University of California, Berkeley, USA  
Dr. Dan Burns, MERL, Cambridge, USA  
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy  
Prof. François E. Cellier, ETH Zürich, Zürich, Switzerland  
Dr. Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany  
Mike Dempsey, Claytex Services Ltd, UK  
Dr. Bernard Dion, Esterelle Technologies, Paris, France  
Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden  
Dr. Hosam Fathy, PennState, University Park, PA, USA  
Prof. Gianni Ferretti, Politecnico di Milano, Italy  
Dr. Rüdiger Franke, ABB AG, Mannheim, Germany  
Prof. Peter Fritzson, Linköping University, Sweden  
Dr. Degang Fu, United Technologies Research Lab, Shanghai, China  
Dr. Rui Gao, Dassault Systèmes Japan, Tokyo, Japan  
Prof. Manfred Hajek, TU Munich, Munich, Germany  
Peter Harman, CyDesign, Coventry, United Kingdom  
Dr. Andreas Heckmann, DLR, Munich, Germany  
Anton Haumer, Technical consultant, St. Andrae-Woerden, Austria  
Dr. Dan Henriksson, Dassault Systèmes, Lund, Sweden  
Prof. Bengt Jacobson, Chalmers Technical University, Gothenburg, Sweden  
Bill Janssen, PARC, Palo Alto, USA  
Jochen Köhler, ZF AG, Friedrichshafen, Germany  
Dr. Christian Kral, Vienna, Austria  
Imke Lisa Krüger, Modelon GmbH, Hamburg, Germany  
Dr. Chris Laughman, MERL, Cambridge, USA  
Prof. Alberto Leva, Politecnico di Milano, Italy  
Kilian Link, Siemens AG, Erlangen, Germany  
Prof. Edward Lee, UC Berkeley, USA  
Dr. Sven-Erik Mattsson, Dassault Systèmes, Lund, Sweden  
Kristin Majetta, Fraunhofer IIS, Dresden, Germany  
Dr. Jakob Mauss, QTronic GmbH, Berlin, Germany  
Dr. Lars Mikelsons, Bosch-Rexroth GmbH, Lohr am Main, Germany  
Prof. Dr.-Ing. Dirk Müller, Aachen, Germany  
Ramine Nikoukhah, Altair Development France, Antony, France  
Prof. Mattias Nyberg, Scania AB, Södertälje, Sweden

Dr. Hans Olsson, Dassault Systèmes, Lund, Sweden  
Prof. Martin Otter, DLR, Germany  
Prof. Peter Pepper, TU Berlin, Berlin, Germany  
Dr. Nicolas Pernet, IFP Energies nouvelles, Rueil-Malmaison, France  
Dr. Adrian Pop, Linköping University, Sweden  
Johan Rhodin, Wolfram Research, Illinois, USA  
Dr. Michael Sasena, LMS, , Ann Arbor, USA  
Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Germany  
Dr. Clemens Schlegel, Schlegel Simulation, Munich, Germany  
Dr. Peter Schneider, Fraunhofer IIS EAS, Dresden, Germany  
Dr. David Smith, Synopsis, Oregon, USA  
Dr. Wilhelm Tegethoff, TLK-Thermo GmbH and TU Braunschweig, Germany  
Dr. Ed Tate, Exa, Livonia, USA  
Eric Thomas, Dassault-Aviation, Paris, France  
Dr. Michael Tiller, Xogeny, Michigan, USA  
Dr. Jakub Tobolar, DLR, Munich, Germany  
Dr. Hubertus Tummescheit, Modelon Inc., West Hartford, USA  
Dr. Andreas Uhlig, ITI GmbH, Dresden, Germany  
Prof. Alfonso Urquía, UNED, Spain  
Dr. Subbarao Varigonda, Cummins, Columbus, USA  
Dr. Stéphane Velut, Lund, Sweden  
Stefan Vorkoetter, MapleSoft, Waterloo, Canada  
Dr. Michael Wetter, LBNL, Berkeley, USA  
Dr. Dirk Zimmer, DLR, Germany

**Local organizers:**

Magnus Gäfvert, Modelon AB  
Jonas Eborn, Modelon AB  
Eva Westin, LCCC, Lund University  
Amelie Rönngard, Anagram



# Contents

<b>Keynote</b>	<b>17</b>
Modelica Evolution - From My Perspective . . . . .	17
<b>Session 1A: FMI 1</b>	<b>27</b>
The Functional Mockup Interface - seen from an industrial perspective . . . . .	27
An FMI-Based Tool for Robust Design of Dynamical Systems . . . . .	35
Simulating Rhapsody SysML Blocks in Hybrid Models with FMI . . . . .	43
Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface . . . . .	53
<b>Session 1B: Automotive Applications 1</b>	<b>63</b>
Model-based Development of Future Small EVs using Modelica . . . . .	63
Modeling of an Electric Axle Drive with Modelica: A Study of Electric Active Dynamics . . . . .	71
Utilizing Object-Oriented Modeling Techniques for Composition of Operational Strategies for Electrified Vehicles . . . . .	79
Thermal shock testing for Engines in Dymola . . . . .	89
<b>Session 1C: Building Energy Applications 1</b>	<b>97</b>
Model-Based Design of Integrative Energy Concepts for Building Quarters using Modelica . . . . .	97
Enhancement of the building simulation software TRNSYS by coupling to the VEPZO model programmed in Modelica . . . . .	107
The Modelica Thermal Model Generation Tool for Automated Creation of a Coupled Airflow, Radiation Model and Wall Model in Modelica . . . . .	115
Modelling long-wave radiation heat exchange for thermal network building simulations at urban scale using Modelica . . . . .	125
<b>Session 1D: Electro-Magnetic Models and Libraries 1</b>	<b>135</b>
Extension of the FundamentalWave Library towards Multi Phase Electric Machine Models . . . . .	135
New Multi Phase Quasi Static Fundamental Wave Electric Machine Models for High Performance Simulations . . . . .	145
The New EDrives Library: A Modular Tool for Engineering of Electric Drives . . . . .	155
Modelica Models for Magnetic Hysteresis, Materials and Transformers . . . . .	165
<b>Session 1E: Modelica Language &amp; Compiler Implementation</b>	<b>173</b>
Custom Annotations: Handling Meta-Information in Modelica . . . . .	173
Modelica extensions for Multi-Mode DAE Systems . . . . .	183
Integrated Debugging of Equation-Based Models . . . . .	195
Making Modelica Applicable for Formal Methods . . . . .	205
<b>Session 2A: FMI 2</b>	<b>213</b>
Implementing stabilized co-simulation of strongly coupled systems using the Functional Mock-up Interface 2.0 . . . . .	213
Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using FMI . . . . .	225
Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems . . . . .	235
Adapting Functional Mockup Units for HLA-compliant Distributed Simulation . . . . .	247
<b>Session 2B: Automotive Applications 2</b>	<b>259</b>
Transmission Modeling in Modelica: A consistent approach for several software development platforms	259
Vectorized single-track model in Modelica for articulated vehicles with arbitrary number of units and axles . . . . .	265
Multibody Model of a Motorbike with a Flexible Swingarm . . . . .	273
Modelling and parameter identification of a semi-active vehicle damper . . . . .	283

<b>Session 2C: Building Energy Applications 2</b>	<b>293</b>
The Modelica HouseModels Library: Presentation and Evaluation of a Room Model with the ASHRAE Standard 140 . . . . .	293
Modelica Library for Building and Low-Voltage Electrical AC and DC Grid Modeling . . . . .	301
Tool coupling for the design and operation of building energy and control systems based on the Functional Mock-up Interface standard . . . . .	311
Coupling occupant behaviour with a building energy model - A FMI application . . . . .	321
<b>Session 2D: Electro-Magnetic Models and Libraries 2</b>	<b>327</b>
Phenomenological Li ion battery modelling in Dymola . . . . .	327
A Modelica Based Lithium Ion Battery Model . . . . .	335
Behavioral Modeling of Power Semiconductors in Modelica . . . . .	343
<b>Session 2E: Modelica Tools 1</b>	<b>353</b>
Verification and Design Exploration through Meta Tool Integration with OpenModelica . . . . .	353
Parallel Model Execution on Many Cores . . . . .	363
A toolchain for Rapid Control Prototyping using Rexroth controllers and open source software . . . . .	371
Modular Multi-Rate and Multi-Method Real-Time Simulation . . . . .	381
<b>Session 3A: Automotive Applications: FMI &amp; HIL</b>	<b>395</b>
Significant Reduction of Validation Efforts for Dynamic Light Functions with FMI for Multi-Domain Integration and Test Platform . . . . .	395
Hardware In The Loop Simulation with Modelica - A Design Tool for Thermal Management Systems . . . . .	401
Integrated Vehicle Thermal Management in Modelica: Overview and Applications . . . . .	409
Virtual Integration for hybrid powertrain development, using FMI and Modelica models . . . . .	419
<b>Session 3B: Fault Handling and Safety Issues in Modelica</b>	<b>427</b>
General fault triggering architecture to trigger model faults in Modelica using a standardized blockset . . . . .	427
Using Fault Augmented Modelica Models for Diagnostics . . . . .	437
From Modelica Models to Fault Diagnosis in Air Handling Units . . . . .	447
Simulation for verification and validation of functional safety . . . . .	455
<b>Session 3C: Novel Modelica Applications and Libraries</b>	<b>465</b>
The Foundation of the DLR RailwayDynamics Library: the Wheel-Rail-Contact . . . . .	465
Human-Nature Interaction in World Modeling with Modelica . . . . .	477
1D/2D Cellular Automata Modeling with Modelica . . . . .	489
Physiolibrary - Modelica library for Physiology . . . . .	499
<b>Session 3D: Electrical Power Systems</b>	<b>507</b>
Modelling of Electrical Power Systems with Dynamic Phasors in Modelica . . . . .	507
Flexible modeling of electrical power systems – the Modelica PowerSystems library . . . . .	515
Implementation of a Multi-Level Power Electronic Inverter Library in Modelica . . . . .	523
Mixed phasor and time domain modelling of AC networks with changeover management . . . . .	533
<b>Session 3E: Modelica Tools 2</b>	<b>543</b>
impact – A Modelica Package Manager . . . . .	543
MoUnit – A Framework for Automatic Modelica Model Testing . . . . .	549
Modeling Parameter Sensitivities via Equation-based Algorithmic Differentiation Techniques: The ADMSL.Electrical.Analog Library . . . . .	557
Modelica Based Parser Generator with Good Error Handling . . . . .	567
<b>Session 4A: Aerospace Applications 1</b>	<b>577</b>
Nonlinear inverse models for the control of satellites with flexible structures . . . . .	577
Modelica Stage Separation Dynamics Modeling for End-to-End Launch Vehicle Trajectory Simulations . . . . .	589
A Modelica Library for Scalable Modelling of Aircraft Environmental Control Systems . . . . .	599
<b>Session 4B: Industrial Equipment</b>	<b>609</b>
Model-Based Energy Recuperation of Multi-Axis Machines . . . . .	609
A Generalized Power-Based Modelica Library with Application to an Industrial Hydraulic Plant . . . . .	617
Physical Design of Hydraulic Valves in Modelica . . . . .	627



<b>Session 4C: Control Applications</b>	<b>637</b>
Exploiting Actuator Limits with Feedforward Control based on Inverse Models . . . . .	637
An FMI-based Framework for State and Parameter Estimation . . . . .	647
Grey-box Building Models for Model Order Reduction and Control . . . . .	657
<b>Session 4D: Thermofluid Systems, Models and Libraries 1</b>	<b>667</b>
Interfacing Models for Thermal Separation Processes with Fluid Property Data from External Sources	667
Development of a Real-Time Fuel Processor Model for HIL Simulation . . . . .	675
ThermoCycle: A Modelica library for the simulation of thermodynamic systems . . . . .	683
<b>Session 4E: Hybrid Systems</b>	<b>693</b>
An Operational Semantics for Hybrid Systems Involving Behavioral Abstraction . . . . .	693
An example of beneficial use of variable-structure modeling to enhance an existing rocket model . . . . .	707
Efficient Monte Carlo simulation of stochastic hybrid systems . . . . .	715
<b>Session 5A: Aerospace Applications 2</b>	<b>727</b>
The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft . . . . .	727
Multi-Level Library of Electrical Machines for Aerospace Applications . . . . .	737
Modelica for large scale aircraft electrical network V&V . . . . .	747
Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters . . . . .	757
<b>Session 5B: Power, Energy &amp; Process Applications 1</b>	<b>767</b>
An Optimization Framework for Dynamic Hybrid Energy Systems . . . . .	767
Industrial application of optimization with Modelica and Optimica using intelligent Python scripting . . . . .	777
Simulation of Smart-Grid Models using Quantization-Based Integration Methods . . . . .	787
On the Simulation of Offshore Oil Facilities at the System Level. . . . .	799
<b>Session 5C: Numerical Aspects of Modelica Tools</b>	<b>809</b>
Parameter Selection in a Combined Cycle Power Plant . . . . .	809
Restarting algorithms for simulation problems with discontinuities . . . . .	819
Discontinuities handled with events in Assimulo . . . . .	827
Noise Generation for Continuous System Simulation . . . . .	837
<b>Session 5D: Thermofluid Systems, Models and Libraries 2</b>	<b>847</b>
A physical solution for solving the zero-flow singularity in static thermal-hydraulics mixing models . . . . .	847
Advanced Hybrid Model for Borefield Heat Exchanger Performance Evaluation, an Implementation in Modelica . . . . .	857
Superheat Control with a Dynamic Inverse Model . . . . .	867
Adsorption energy systems library - Modeling adsorption based chillers, heat pumps, thermal storages and desiccant systems . . . . .	875
<b>Session 5E: Modelica Tools 3</b>	<b>885</b>
A new Implementation of the N-D Lookup Tables . . . . .	885
Remarks on the Implementation of the Modelica Standard Tables . . . . .	893
The DLR Visualization Library - Recent development and applications . . . . .	899
Automated Modelica Package Generation of Parameterized Multibody Systems in CATIA . . . . .	913
<b>Session 6A: Mechanical Systems</b>	<b>923</b>
Modelling elastomer buffers with DyMoRail . . . . .	923
A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces . . . . .	929
The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure - Modal Reduction Method in Modelica . . . . .	939
Simulating Collisions within the Modelica MultiBody library . . . . .	949
<b>Session 6B: Power, Energy &amp; Process Applications 2</b>	<b>959</b>
Short-term production planning for district heating networks with JModelica.org . . . . .	959
Modelling the system dynamics of islanding asynchronous generators . . . . .	969
Hybrid Energy System Modeling in Modelica . . . . .	979
Dynamic Modeling of Small Modular Nuclear Reactors using MoDSim . . . . .	989

<b>Session 6C: Optimization Applications and Methods</b>	<b>999</b>
Modified Multiple Shooting Combined with Collocation Method in JModelica.org with Symbolic Calculations . . . . .	999
DOML – a Compiler Environment for Dynamic Optimization Supporting Multiple Solvers . . . . .	1007
Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C . . . . .	1017
Symbolic Transformations of Dynamic Optimization Problems . . . . .	1027
<b>Session 6D: Thermal Power Processes</b>	<b>1037</b>
Modelling a Lignite Power Plant in Modelica to Evaluate the Effects of Dynamic Operation and Offering Grid Services . . . . .	1037
Use of External Fluid Property Code in Modelica for Modelling of a Pre-combustion CO <sub>2</sub> Capture Process Involving Multi-Component, Two-Phase Fluids . . . . .	1047
Dynamic modelling of a parabolic trough solar power plant . . . . .	1057
Testing Power Plant Control Systems in Modelica . . . . .	1067
<b>Session 6E: Web-related Modelica Tools</b>	<b>1073</b>
Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis . . . . .	1073
recon – Web and network friendly simulation data formats . . . . .	1081
IDOS - (also) a Web Based Tool for Calibrating Modelica Models . . . . .	1095
Client-side Modelica powered by Python or JavaScript . . . . .	1105
<b>Poster Session</b>	<b>1113</b>
Dynamic modelling of a Condenser with the ThermoSysPro Library . . . . .	1113
Wavelet Library for Modelica . . . . .	1123
Model-based Verification and Optimization of Batteries for Mobile Power Applications . . . . .	1131
Systems Physics Library . . . . .	1137
Implementation of the Omni Vehicle Dynamics on Modelica . . . . .	1143
Control and Characteristic Map Generation of Permanent Magnet Synchronous Machines and Induction Machines with Squirrel Cage . . . . .	1151
BuildSysPro: a Modelica library for modelling buildings and energy systems . . . . .	1161
Efficient Numerical Integration of Dynamical Systems based on Structural-Algebraic Regularization avoiding State Selection . . . . .	1171
Symbolic Initialization of Over-determined Higher-index Models . . . . .	1179
Proposal for standardization of Heat Transfer Modelling in NewThermal Library . . . . .	1189
A Modelica Power System Component Library for Model Validation and Parameter Identification . . . . .	1195
Modelica Model for the youBot Manipulator . . . . .	1205
Equation based parallelization of Modelica models . . . . .	1213
Simulation of 2-dimensional flows in Modelica with the Cascaded Digital Lattice Boltzmann Method . . . . .	1221
FORM-L: A MODELICA Extension for Properties Modelling Illustrated on a Practical Example . . . . .	1227
Statecharts as a Means to Control Plant Models in LMS Imagine.Lab AMESim . . . . .	1237
Integration of OpenModelica in Ptolemy II . . . . .	1247
Extending JGrafchart with Support for FMI for Co-Simulation . . . . .	1257
Development of Custom Workflows for Simulation and Analysis of Functional Mock-up Units . . . . .	1265
A Medium Model for the Refrigerant Propane for Fast and Accurate Dynamic Simulations . . . . .	1271
Consistent Simulation Environment with FMI based Tool Chain . . . . .	1277
A MATLAB to Modelica Translator . . . . .	1285
Setting up a framework for model predictive control with moving horizon state estimation using JModelica . . . . .	1295
<b>Exhibitors</b>	<b>1305</b>
BAUSCH-GALL . . . . .	1305
CENIT AG . . . . .	1305
Claytex . . . . .	1305
Concurrent Real-Time . . . . .	1305
CyDesign Labs . . . . .	1306
D2T . . . . .	1306
Dassault Systèmes . . . . .	1306
ETAS . . . . .	1306
Esterel Technologies . . . . .	1307

IPG Automotive . . . . .	1307
ITI . . . . .	1307
LMS International . . . . .	1307
Maplesoft . . . . .	1308
Modelon AB . . . . .	1308
OpenModelica . . . . .	1308
Schlegel Simulation . . . . .	1308
Wolfram Research . . . . .	1309
XRG Simulation GmbH . . . . .	1309



## Author Index

Åberg, Erik	675	Ceriani, Nicola	787
Acquatella, Paul	589	Cetiner, Sacit	989
Ahle, Elmar	27	Chakirov, Roustiam	1205
Åkesson, Johan	35, 657, 777, 809, 827, 1027, 1277	Christmann, Markus	757
Alkov, Ilja	617	Colonna, Piero	1047
Amouroux, Edouard	321	Constantin, Ana	293
Andersson, Bengt-Arne	1277	Costes, Joris	799
Andersson, Christian	827	De Coninck, Roel	657, 1295
Andersson, Daniel	1277	De Kleer, Johan	205, 353, 437
Andersson, Niklas	809	Del Hoyo Arce, Itzal	1189
Andres, Markus	79, 343, 885, 1151	Denz, Patrick	343
Arévalo, Carmen	819	Desideri, Adriano	683
Asghar, Adeel	195	Diaz, Gonzalo	89
Asher, Greg	507	Diehl, Stephan	885
Awad, Hasan Flaih	71	Diekmann, Robin	617
Axelsson, Karin	675	Dietl, Karin	777, 809
Bachmann, Bernhard	195, 1017, 1179	Dolanc, Gregor	675
Baeuml, Thomas	1221	Drenth, Edo	1277
Bals, Johann	577, 1123	Driesen, Johan	301
Baltzer, Sidney	401	Du, Wenbo	767
Bapty, Ted	235, 353	Dumont, Elisabeth	923, 1137
Bardow, André	875	Durak, Umut	247
Batteh, John	235, 409, 989	Duval, Laurent	225
Bau, Uwe	875	Dwiputra, Rhama	1205
Baudette, Maxime	1195	Eborn, Jonas	675
Baumgartner, Daniel	913	Eckstein, Lutz	401
Bell, Ian	683	Edvardsen, Håkon Molland	969
Bellmann, Tobias	899	Egaña, Imanol	757
Belmon, Lionel	419	El Hefni, Baligh	847, 1113
Ben Gaid, Mongi	225	Elmqvist, Hilding	17, 53, 173, 183, 363, 715
Ben Khaled, Abir	225	Elsheikh, Atiyah	557
Benveniste, Albert	715	Febres, Jesús	447
Bergero, Federico	787	Feldman, Yishai A.	43
Bergmann, Julien	707	Ferretti, Gianni	273
Berndt, André	1037	Fleps-Dezasse, Michael	53, 283
Berntorp, Karl	1027	Floros, Xenofon	787
Berthoux, Vincent	1237	Folie, Michael	395
Bertsch, Christian	27	Franke, Marco	609, 1131
Beutlich, Thomas	893	Franke, Rüdiger	515, 1105
Binder, William	979	Fredriksson, Emil	827
Bliudze, Simon	693	Frimberger, Johannes	395
Bobrow, Daniel	205, 353, 437	Fritzson, Peter	195, 353, 477, 549, 567, 1247, 1285
Bödrich, Thomas	165	Fröjd, Karin	675
Bogodorova, Tetiana	1195	Fuchs, Marcus	125
Boman, Katarina	959	Fugate, David	989
Bonifay, Julien	1067	Führer, Claus	819
Bonvini, Marco	647	Furic, Sébastien	693, 1237
Bouissou, Marc	715	Galindo, Eduardo	89
Bouskela, Daniel	847, 1113	Gall, Leo	1067
Bozhko, Serhiy	507, 523, 737	Gallardo Yances, Stephanie	777, 809
Braun, Willi	195, 1017	Gao, Jianbo	1123
Brembeck, Jonathan	53	Garcia, Humberto	767, 979
Buggert, Matthias	1067	Geletu, Abebe	999
Casella, Francesco	195, 667, 787, 1047, 1179	Gerada, Chris	523, 737
Castro, Rodrigo	477	Gerl, Johannes	335
Cellier, François	477, 787	Ghidaglia, Jean-Michel	799

Giangrande, Paolo	523, 737, 757	Kral, Christian	135, 145, 155
Gissing, Jörg	401	Krüger, Imke	335
Gohl, Jesse	235, 409, 1265	Kübler, Svea	107
Gomez Esperon, Daniel	707	Kuebler, Michael	259
Gräber, Manuel	637, 867, 875	Küfen, Jörg	401
Graf, Stefan	875	Kuhn, Martin R.	747
Greenberg, Lev	43	Kühnelt, Helmut	1221
Grün, Gunnar	115	Kulhánek, Tomáš	499
Gu, Xin	939	Kulshreshtha, Kshitij	1017
Gubsch, Ines	949, 1213	Kurtoglu, Tolga	353, 437
Hale, Richard	989	Kurzbach, Gerd	173, 893
Hämmerle, Sebastian	79	Kurzeck, Bernhard	465
Haradji, Yvon	321	Laine, Leo	265
Harman, Peter	1081	Lanzerath, Franz	875
Haumer, Anton	135, 145, 155	Larsson, Per Ola	959
Heckmann, Andreas	465	Lattmann, Zsolt	235, 353
Hellerer, Matthias	899	Lauster, Moritz	125
Helsen, Lieve	657, 857, 1295	Lazutkin, Evgeny	999
Henningsson, Maria	35	Lemort, Vincent	683
Hill, Christopher	523, 737, 757	Lemouedda, Abdellah	115
Hillmann, Claudio	549, 939	Lerche, Christian	97
Hirano, Yutaka	63	Leva, Alberto	489, 533
Hofmann, Andreas	949	Li, Pu	999
Hopfgarten, Siegbert	999	Lichius, Thomas	401
Huebel, Moritz	1037	Lindsay, Amy	1161
Inoue, Shintaro	63	Link, Kilian	777, 809, 1067
Jacobson, Bengt	265	López Pérez, Susana	1189
Jahangiri, Pooyan	1271	Magnusson, Fredrik	657, 1027
Jahanzeb, Mohammad	1285	Maré, Jean-Charles	757
Janczyk, Leonard	335	Matei, Ion	437
Janssen, Bill	205, 353, 437	Mateják, Marek	499
Jeck, Peter	401	Mattsson, Sven Erik	183, 363, 381
Ježek, Filip	499	Maurer, Werner	923, 1137
Ji, Yang	747, 1123	Mehlhase, Alexandra	707
Johansson, Krister	1277	Meinke, Sebastian	1037
Johnsson, Anna	777	Menager, Nils	371
Johnsson, Charlotta	1257	Merkle, Marcel	707
Jordan, Philip	599	Mikelsons, Lars	371, 455, 949
Juhasz, Tamas	609, 1131	Minhas, Raj	437
Kaemmerlen, Aurelie	1161	Mirzaei, Mana	1247
Kaiser, Ingo	465	Modrow, Nils	335
Karsai, Gabor	235	Mohammadi, Fatemeh	819
Keane, Marcus	447	Motesharrei, Safa	477
Keck, Alexander	465	Muguerra, Philippe	799
Kehrer, Christian	97	Müller, Dirk	125, 293, 1271
Keßler, Marco	79, 1151	Neema, Himanshu	235
Kennel, Matthias	609	Neema, Sandeep	235, 353
Kennel, Ralph	1123	Nguyen, Thuy	1227
King, Julian	259	Nielsen, Keld Lund	799
Klasing, Freerk	1271	Nilsson, Bernt	809
Klenk, Matthew	205, 353	Norrefeldt, Victor	107, 115
Klöckner, Andreas	727, 837	Nouidui, Thierry Stephane	311
Kofman, Ernesto	787	Ochel, Lennart	195, 1179
Kofránek, Jiří	499	Oestersötebier, Felix	929
Köhler, Jochen	259	Oğüztüzün, Halit	247
Köhler, Jürgen	867	Olofsson, Björn	1027
Kosenko, Ivan	1143	Olsson, Hans	363
Koutsoukos, Xenofon	353	Österholm, Robert	1057

Osvaldsson, Erik	675	Sterling, Raymond	447
Ota, Junya	63	Strach, Mareike	549
Otter, Martin	53, 173, 183, 381, 715	Streblow, Rita	125, 293, 1271
Palachi, Eldad	43	Struss, Peter	447
Palanisamy, Arunkumar	567, 1285	Su, Zhou	455
Pålsson, Jens	675, 1057	Sundström, Peter	265
Paredis, Christiaan	767, 979	Sureshkumar, Chandrasekar	235, 409, 627, 1265
Parildar, Hakan	533	Sztipanovits, Janos	235
Pathak, Arnav	115	Tarnawski, Tomasz	1007, 1095
Pfeiffer, Andreas	53, 913	Taylan, Koray	247
Picard, Damien	857	Tegethoff, Wilhelm	867
Picarelli, Alessandro	89, 327	Teichmann, Jens	125
Pitzer, Johannes	283	Theorin, Alfred	1257
Plessis, Gilles	321, 1161	Thiele, Bernhard	381
Pop, Adrian	195, 353, 567	Thielen, Niklas	875
Prassler, Erwin	1205	Thomas, Philipp	939
Pregelj, Bostjan	675	Tiller, Michael	543, 989, 1073, 1081
Privitzer, Pavol	499	Tobolar, Jakub	283
Pytlak, Radosław	1007, 1095	Törmänen, Mikael	1277
Qualls, Lou	989	Torstensson, Ivar	675, 1277
Quoilin, Sylvain	683	Trächtler, Ansgar	929
Regula, Gergely	757	Trapp, Carsten	667, 1047
Reiner, Matthias	577, 589	Tummescheit, Hubertus	35, 235, 627
Remmen, Peter	125	Uddin, Kotub	327
Rettig, Frank	71	Unger, René	97
Riou, Xavier	799	Urquia, Alfonso	489
Rivas, Jorge	477	van der Linden, Franciscus	427, 757, 837
Rogovchenko-Buffoni, Lena	1247	van der Stelt, Teus	1047
Rossi, Andrea	273	Van Roy, Juan	301
Ruge, Vitalij	1017	Vande Cavey, Mats	1295
Saarinen, Linn	959	Vanfretti, Luigi	1195
Sabir, Umbreen	447	Varchmin, Andreas	867
Saha, Bhaskar	353, 437	Vayatis, Nicolas	799
Salenbien, Robbe	301	Velut, Stéphane	777, 959
Samlaus, Roland	549, 939	Viel, Antoine	213
Sangi, Roozbeh	1271	Wagner, Loïc	1237
Sanz, Victorino	489	Walther, Andrea	1017
Saut, Jean-Philippe	799	Walther, Marcus	1213
Scaglioni, Bruno	273	Wang, Peng	929
Schlegel, Clemens	757	Waurich, Volker	1213
Schlegel, Florian	899	Weidemann, Dirk	617
Schmitt, Thomas	343, 1151	Wellner, Kai	667
Schmitz, Gerhard	599, 667	Wernersson, Karl	53
Schmucker, Ulrich	609, 1131	Wetter, Michael	311, 647
Schnabel, Uwe	893	Wiesmann, Hansjürg	515
Schneider, Stefan-Alexander	395	Wihlfahrt, Urs	939
Scholz, Lena	1171	Windahl, Johan	959
Schreiber, Heike	875	Winkler, Dietmar	543, 969
Schubert, Christian	949, 1213	Wöhrnschimmel, Reinhard	135
Schulmeister, Ulrich	27	Worschech, Niklas	371
Schwan, Torsten	97	Wronski, Jorrit	683
Šilar, Jan	499	Yang, Tao	507
Simon, Daniel	225	Yilmaz, Faruk	247
Sjölund, Martin	195, 567, 1285	Zakharov, Alexey	1205
Smetana, Tomas	71	Zimmer, Dirk	173, 837
Sohn, Michael D.	647	Ziske, Johannes	165
Sommer, Torsten	885		
Steinbrecher, Andreas	1171		





# Modelica Evolution – From My Perspective

Hilding Elmqvist  
Dassault Systèmes  
Ideon Science Park, SE-223 70 Lund, Sweden  
Hilding.Elmqvist@3DS.com

## Abstract

This paper intends to tell the story of Modelica ([www.Modelica.org](http://www.Modelica.org)) from the author's perspective. It is a fantastic saga that for me started in April 1976. The saga includes studying the needs, the original idea, the development of a solution, waiting for mature hardware technology, a start-up company, a fantastic collaboration, an automotive company caring for its software supplier, how to get momentum by standards collaboration, forming the right team, and the magic phone call from the right company.

*Keywords:* Simnon; Dymola; Modelica; Physical Modeling; DAE; Modelica Association

## 1 Introduction

In order to describe Modelica evolution it is important to start by describing Dymola, Dynamic Modeling Language, since many of the original features of Modelica are taken from Dymola.

The paper then describes forming the company Dynasim and the important initial research collaborations. The initial Modelica design work and the Modelica community are then discussed. The paper ends with a discussion about Modelica future. Most of the material is presented in chronological order.

For an overview of other simulation tools, see (Åström, et.al., 1998).

I propose that other key contributors to Modelica evolution, Martin Otter, Peter Fritzson and many others, write similar papers from their perspectives in particular since this paper has a focus on the work before Modelica and gives my perspective.

## 2 Dynamic Modeling Language

When looking for a master thesis project in autumn 1971, the Automatic Control Department, Lund Institute of Technology wanted to develop a simulation program where the user could input the models using

mathematical expressions. I was just studying compiler technology, so I considered this project perfect for me. The result was Simnon (SIMulation of NON-linear systems): (Elmqvist, 1975, 1977a). A novel feature was that a mixture of continuous time and discrete time systems could be simulated, i.e. perfect for simulation of sampled data control systems which were becoming very important at that time.

However, I was not satisfied with the approach as a basis for my PhD thesis, since it was based on submodels with inputs and outputs. I therefore wanted to investigate the fundamentals of modeling which my professor Karl Johan Åström supported. In April 1976, I was studying a report with about 200 pages about a drum-boiler model written by Sture Lindahl (Lindahl, 1976). The report was very systematic, showing how the equations were organized for the drum, the super heaters, the turbines, etc., i.e. the approach was object oriented. The modeling started by stating the relationships as general equations. Then the derivations were shown how to put them into state space form in order to be able to simulate using Simnon. These derivations involved manual symbolic manipulation of individual equations to solve for the unknown, solving linear systems of equations (see excerpt from report below), unrolling Newton's algorithm a fix number of times to solve nonlinear systems of equations, and differentiating certain equations when there were constraints between differentiated variables (nowadays called index reduction).

The image shows a handwritten mathematical derivation from a report. It includes two equations, (9.1) and (9.2), with various annotations in blue and red ink. Equation (9.1) is a mass balance equation for a steam-water mixture. Equation (9.2) is a similar equation. A red box highlights the step where equation (9.2) is multiplied by  $h_{dw8}$  and then equation (9.1) is subtracted from it.

$$\frac{d}{dt} [c_{dm8} m_{dm8} T_{dm8} + (V_{dl8} - V_{ds8}) \rho_{dw8} h_{dw8} + (V_{ds2} + V_{ds8}) \rho_{ds2} h_{ds2}] =$$
$$= q_{dm8} + w_{dl5} h_{dl4} - w_{ps1} h_{ds2} - w_{dw8} h_{dw8} \quad (9.1)$$

The mass balance the steam-water mixture in the risers and the steam in the drum becomes:

$$\frac{d}{dt} [(V_{dl8} - V_{ds8}) \rho_{dw8} + (V_{ds2} + V_{ds8}) \rho_{ds2}] =$$
$$= w_{dl5} - w_{ps1} - w_{dw8} \quad (9.2)$$

Multiplication of equation (9.2) by  $h_{dw8}$  and subtraction from equation (9.1) gives:

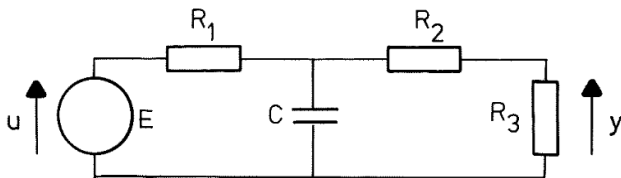
Just before Easter 1976, I realized what I needed to do: design a new language allowing general equations, having a class concept for object oriented modeling (I was used to Simula so that was natural) and having a structured feature to describe interaction between submodels (called cut). The language was called Dymola for DYNAMIC MODELING LANGUAGE (Elmqvist, 1977a, 1978, 1979a, 1979b).

An example in the Dymola language of an electrical component is shown below. It describes a pin A characterized by an across variable  $V_a$  and a through variable  $I$ . Pin B has correspondingly  $V_b$ . The through variable of B is also  $I$  but with a preceding minus sign showing that the currents of pins A and B sum to zero. In Dymola, all variables were of real type. The der operator is the same as in Modelica. The path concept was introduced in order to allow operators for series-, parallel- and loop-connections. Such a concept is not used in Modelica since connections are typically drawn graphically.

```

model type capacitor
  cut A (Va / I) B (Vb / -I)
  main cut C [A B]
  main path P <A - B>
  local v
  parameter C
  V = Va - Vb
  C*der(V) = I
end
    
```

Consider the following electrical circuit:



Using a library of “model types”: resistor, capacitor, voltage and Common, the circuit could be described as follows;

```

model Network
  submodel(resistor) R1 R2 R3
  submodel(capacitor) C
  submodel(voltage) E
  submodel Common
  input u
  output y
  connect Common to E to R1 to (C par (R2 to R3)) to Common
  E.V = u
  y = R3.Va
end
    
```

The “to” operator means series connection and “par” means parallel connection.

The trouble was then to figure out how to simulate such a model. I had used the symbolic package REDUCE and started to write symbolic manipulations in LISP. But how to know what variable to

solve for in each equation? The naïve approach is: look for equations with only one unknown, mark those as known and repeat. The problem is that there can be mutual dependencies, i.e. algebraic loops for which this approach does not work. And mutual dependencies were normal in the way Lindahl did his modeling, so the language could not be restricted to avoid algebraic loops. I invented my own algorithms using binary incidence matrices. However, these were  $O(n^3)$ , so they would not scale.

At that time one had to go to the university library to get help in doing a computerized literature search (no Alta Vista or Google being available). I found that graph theoretical algorithms were  $O(n)$ . The graphs are bipartite, i.e. consists of two types of nodes corresponding to equations and variables. An edge between a variable and an equation means that the variable appears in that equation. A report from the University of Umeå analyzed several algorithms for Assignment or matching which corresponded to finding out what variable to solve for in each equation. Furthermore, Tarjans algorithm would find strongly connected components in a directed graph corresponding to minimal systems of simultaneous equations and sort them into a sequence suitable for solving one system at a time, i.e. the incidence matrix becomes block lower triangular (BLT).

Lindahl sometimes differentiated equations when potential states were constrained. The Jacobian is then always singular. I designed an algorithm to find what equations to differentiate and tested it successfully on some small examples. Fortunately, Costas Pantelides and Sven Erik Mattsson later figured out suitable algorithms, (Pantelides (1988), (Mattsson, Söderlind, 1993).

To test the language ideas and the algorithms, I wrote a program in the object oriented programming language Simula, which is included in my thesis, (Elmqvist, 1978). I tested it successfully on a few examples from different domains: electronics, mechanical, thermodynamics, and electrical power distribution. As an example, the program found one system of equations for solving accelerations and forces of mechanical systems corresponding to the need to invert the mass matrix. The drum boiler model could now be formulated in 15 pages using original equations defined only once for each class and no need for manual symbolic manipulation of the equations. The total model had about 250 non-trivial equations and 11 systems of simultaneous equations.

But there was one big problem: lack of computer memory for symbolic manipulation. I was barely able to translate the drum boiler model on a Univac 1108 computer having only 64 kWords of 36 bits of

memory accessible for the user and the garbage collector had to work hard. So the hardware technology at that time was not mature enough for realistic industrial problems.

After my thesis in May 1978 (Elmqvist, 1978), I spent one year as a post doc at Computer Science Department at Stanford University. I took all the courses in languages and compilers given by John Hennessy (who later became President of Stanford University). Working in a compiler project regarding parametric types, (Hennessy, Elmqvist, 1982) was a very useful experience.

I then returned to the Automatic Control department for a project on Languages for Implementation of Control Systems (LICS), (Elmqvist, 1985). In addition to language elements, we also considered the relations to graphics and were able to acquire a mouse. That gave valuable knowhow which later became meta comments in Dymola and adopted in Modelica as graphical annotations. In order to achieve sufficient speed for navigating in hierarchical structures, we developed special raster graphics hardware and a font generation program. The LICS program was developed on a VAX-11 computer and ported to an Apollo DN 600 workstation and demonstrated at the second IEEE Computer Aided Control Systems Design Symposium in 1983 at MIT with high interest since it showed several novel interaction principles such as “information zooming”.

During this project, we also translated the Dymola program from Simula to Pascal since there was no Simula compiler on the VAX. This enabled handling larger models due to the linear address space of the VAX. Francois Cellier got a copy of the Pascal version. The importance will be clear later.

The LICS project got an industrial follow-up since I worked for the company SattControl 1984-1992. We developed a distributed control system called SattLine (Elmqvist, 1991a, 1991b, 1992). This experience later influenced the design of hybrid features of Modelica.

### 3 The Company Dynasim

In August 1991, an important thing happened. I was working in Toronto for two years and attended a seminar at the university by a colleague of Francois Cellier. Francois had moved to Tucson, Arizona and we had lost contact. During our renewed contacts I learnt that Francois was writing a book: Continuous Systems Modeling (Cellier, 1991). I got the manuscript and was overwhelmed! It described Dymola and used Dymola for modeling of many examples.

I also met Peter Fritzon in the summer 1991 at a computer language conference in Toronto and became aware of his work on designing a modeling language, ObjectMath, which combined features of Computer Algebra languages and Object Oriented Programming. I successfully converted some ObjectMath models to Dymola.

These were two of the tipping points that made me resume the work on Dymola since I understood that there was finally an interest in equation based modeling. The third tipping point was the release of Windows 3.0 since the memory barrier of 640 kBytes linear memory was removed.

Cellier and I started to work together and we wrote a paper for a CACE conference in Napa, California (Cellier, Elmqvist, 1992) which later became an article in IEEE Control Systems (Cellier, Elmqvist, 1993). I continued to make development on the Pascal version of Dymola for some time and used the p2c (Pascal to C) translator to facilitate using a C compiler. Sven Erik Mattsson had given me a copy of the article by Costas Pantelides about index reduction, (Pantelides, 1988). I incorporated this in Dymola for the paper.

At the same conference, Sven Erik presented the dummy derivative method including automatic state selection after index reduction based on Pantelides algorithm (Mattsson, 1992).

In Napa, I met Georg Grübel, DLR (Deutsches Zentrum für Luft- und Raumfahrt) and learnt that Cellier would spend the summer 1992 at DLR in Oberpfaffenhofen, Germany. It was decided that Cellier would introduce the Dymola technology at DLR. Martin Otter immediately got hooked and started to implement a library of MultiBody components which was later published in (Otter, et.al., 1996). Francois remembers: “Four weeks into my visit, Martin came to my office on Monday morning, telling me that he now could throw away his program "myRobot" that he had previously developed for the simulation of tree-structured robots. Over the weekend, he had re-implemented his robot model in Dymola, which turned out to be much more elegant and flexible.”

I moved back to Sweden and started the company Dynasim in August 1992. DLR realized the potential of the Dymola technology but also the needs to extend the Dymola language and the tool. For example, Dymola did not have **matrix support** then which is essential for MultiBody modeling. Fortunately, DLR could support further development financially which gave Dynasim important initial funding. Martin Otter and I have worked in close collaboration since then. We met for the first time at DLR Oberpfaffenhofen in August 1993 after one year of collaboration via e-mail.

We made a design for **modeling hybrid systems** in order to be able to model power electronics and phenomena such as friction, (Cellier, et.al., 1993) and (Otter, 1994). The concepts of instantaneous equations and event iterations were then introduced.

In July 1993, Dag Brück was employed especially to develop a **graphical user interface** enabling creating components and connecting them by drag and drop. Dag had worked in the Omola project at the Automatic Control Department, Lund.

A missing piece of the technology was to handle that the blocks corresponding to systems of simultaneous equations often got large but sparse. Tearing is a static technique to handle this problem. Our first attempt was to help the translator by introducing hints in the form of a residue operator, (Elmqvist, Otter, 1994). Shortly afterwards, Martin Otter and myself figured out an algorithm to automatically perform tearing, and the algorithm was introduced in Dymola. Dymola would then, for example, for a tree structured multibody model find a large linear system of equations involving accelerations, forces and torques and discover that it could be reduced to inverting a linear matrix of size equal to the number of degrees of freedom.

The efficiency of the code could be enhanced if structural and symbolic processing is made not only on the model equations but also on the discretization equations of the numerical integrator. We called it **inline integration**, (Elmqvist, et.al., 1995). This became very important for hardware-in-the-loop simulation. For example, this technique allows efficient use of implicit methods for multibody models since it combines inverting the mass matrix and solving the non-linear systems of equations.

The **early adopters** of this equation and object based modeling technology can be seen from the Dymola license numbers:

- 101: Francois Cellier, University of Arizona
- 102: Martin Otter, DLR
- 103: Rafael Huber, Dept. Ing. Sistemas, ESAIL, Barcelona, Spain
- 104: Sebastian Dormido, Depto. De Informatica y Automatica, UNED, Spanien
- 105: Michael Grimsberg, Lund Institute of Technology
- 106: Dieter Kraft, Dept of Mechanical Engineering, University of München
- 107: Jochen Seibold, Rechenzentrum University of Stuttgart

Ingrid Bausch-Gall early understood the potential of Dymola and Bausch-Gall GmbH became **distributor** in German speaking countries in September

1995. This was very important for growing the interest in the technology in Germany.

Bengt Jacobson, Machine and Vehicle Design, Chalmers Technical University was among the early adopters and started to use Dymola for powertrain development in a collaborative project with Volvo and Saab in 1995.

Jan Tuszynski at Sydkraft Konsult in Malmö, Sweden deployed Dymola for thermal power systems in 1995.

Sebastian Dormido organized a workshop in Avila, Spain summer 1996. Francois Cellier and I presented the principles of object-oriented equation based modeling.

Akira Ohata and Yutaka Hirano, Toyota Motor Corporation heard about Dymola during a presentation in Japan by Georg Grübel, DLR. They got Dymola licenses in middle of 1996 and started to use it for the **Toyota Prius development**. The multi-engineering modeling capability was necessary for the hybrid car development. However, they realized that the maturity of the Dymola software was not sufficient for large scale deployment and provided funding during 2 years starting end of 1997 regarding general GUI enhancements, GUI for Modelica, symbolic solver capabilities, etc. This was an essential funding allowing for expansion of Dynasim. In the end of 1998, Toyota Techno Service Corporation became distributor of Dymola in Japan with Takashi Matsuba and Rui Gao as technical support. In 2007, Rui Gao came to work for Dassault Systèmes.

Martin Otter visited Dynasim for two months in September and October 1996 and gave a lecture series on object oriented modeling of mechanical systems at the Automatic Control department at Lund Institute of Technology. We worked in particular on real-time simulation of hybrid systems such as automatic gearboxes. Martin was the main author of the **Dymola standard library** containing Blocks, Electrical, Rotational, and MultiBody components.

Hans-Jürg Wiessmann and Bernhard Bachmann, ABB, Baden-Dättwil in Switzerland became users of Dymola in 1996 for **power electronics** modeling and HILS (Hardware-In-the-Loop Simulation). Bernhard spent 4 weeks at Dynasim in March 1998. He then proposed a simple state selection algorithm which was used in Dymola before the dummy derivative algorithm of Sven Erik Mattsson was included.

Peter Beater, then at University of Paderborn, started to develop a **Hydraulics library** beginning of 1997. This was later converted to Modelica and sold to Modelon in 2006.

Rüdiger Franke (then at University of Ilmenau) started to use Dymola with partial Modelica support in 1997 for optimization of thermodynamic systems.

I started to completely rewrite the **symbolic engine** of Dymola in July 1997. Until then the old version converted to C from my thesis had been used. I used C++ Standard Template Library successfully on several hundred thousands of equations and Dag Brück implemented counted pointers to handle allocation and de-allocation of memory automatically.

Martin Otter held his first modeling course based on Dymola at Technical University of Munich for students of electrical and mechanical engineering in the year 1997.

Mike Dempsey (then at Rover) started to evaluate Dymola in 1998 for powertrain modeling.

Sven Erik Mattsson was employed in August 1998. He had a perfect background from the Omola project (Andersson, 1994) since he implemented the symbolic engine of OmSim.

Hans Olsson was employed in January 1999 and had a complementing background regarding numeric and GODESS solvers. So now, the **dream team** was formed with Dag, Sven Erik, Hans and me together with collaboration with Martin Otter.



Sven Erik, Dag, Hilding, Hans

Roger Larsson was utilized as a consultant from June 2004 to accelerate building the business side of Dynasim.

## 4 Modelica Language Design

In spring 1996, I had lunch with Prof Karl Johan Åström and Sven Erik Mattsson from the Automatic Control department, Lund University. Sven Erik was the project leader of the Omola project (Andersson, 1994). We discussed the need for unification of our efforts on Omola and Dymola.

In May 1996, I attended a workshop in Brussels regarding simulation methodologies organized by Hans Vangheluwe. I then, for the first time, met Per Sahlin, Bris Data AB, Stockholm (later renamed to Equa AB) and Alexandre Jeandel, Gas de France, Paris who both were working on equation oriented modeling languages: NMF and Allan respectively.

I then realized that it was time for a **global unified language design initiative**. Modeling requires reuse of stored knowledge, i.e. there must be a standard language. It does not make sense that various tool vendors invent their own language and that a new language is created for every Ph.D. thesis on modeling.

So, I initially phoned Sven Erik Mattsson, Martin Otter, Per Sahlin and Alexander Jeandel to discuss starting a collaborative language design effort. We agreed to meet in Lund on September 2-4, 1996. The participants are seen below, from left to right: Martin Otter, Alexandre Jeandel, Per Sahlin, Sven Erik Mattsson, Bernt Nilsson and me. Dag Brück participated partially and took the photo.



The picture below shows participants of the 75<sup>th</sup> Modelica design meeting in Lund May 2012.



You might wonder about the name Modelica. The name was invented during one of the lunches of the first meeting. I don't recall who said it first. We quickly checked the usage in Alta Vista. It is used in Spanish for a role model.

Sven Erik Mattsson organized a COSY workshop in Lund the week after with about 20 participants.

Our language design ideas were then presented, i.e. it got some immediate attention. We met Costas Pantelides at this workshop and convinced him to arrange the 3<sup>rd</sup> design meeting in London. One of the goals was to get in closer contact and cooperate together. Unfortunately, Costas decided later to not join the Modelica development and continued with his own modeling language, gPROMS.

The language design effort initially got a small travel funding from the ESPRIT project "Simulation in Europe Basic Research Working Group (SiE-WG)". It was later organized as the EuroSim Modelica Technical Committee 1 and the Technical Chapter on Modelica within Society for Computer Simulation. I was the chairman for 3 years and then handed over to Martin Otter, the present chairman.

In the year 2000, the non-profit Modelica Association was formed in Linköping, Sweden to formalize the continually evolving Modelica language and the development of the free Modelica Standard Library.

**Modelica 1.0** was released after one year design in September 1997. It had classes, inheritance, class parameters (replaceable classes and components) equations, blocks, connectors, functions, etc. I remember an intense debate at the release meeting when Peter Fritzson, who joined the Modelica effort early 1997, argued for including functions and I argued to wait until the next release. Thank you Peter for pushing. Hybrid modeling features were included in version 1.1 released in December 1998.

Dag Brück started to implement a Modelica compiler immediately after the first design meeting. It initially output Dymola code in order that the Modelica models could be simulated. Dymola has been used as a platform for the design group to test all features during Modelica evolution. The first version of **Dymola which officially supported Modelica** was version 4.0 released July 1, 1999. We worked all night until 6.10 A.M in the morning since we wanted it to be released first half of 1999 (in at least some part of the world). The meaning of Dymola then had to be changed to DYNAMIC MODELING LABORATORY. It also had a conversion option to convert Dymola models and packages to Modelica. Martin Otter was the main author of the **Modelica Standard Library (MSL)**. Later, many people contributed to the development of MSL and the current version of MSL has Modelica models from about 30 people.

Michael Tiller, then at Ford, became aware of Dymola (the language) but got really interested in Modelica due to the open specification. He became the first Dymola/Modelica power user in beginning of 1999 and helped us test our Modelica translator. He also wrote the first book about Modelica: Introduction to Physical Modeling with Modelica, (Tiller,

2001). We provided a demo version of Dymola which was shipped with the book. Ford was also interested in converting Adams models, so we made a translator from adm-files to Modelica MultiBody. Andreas Möller (now my son-in-law) developed a supporting Modelica library. To handle these large models, we had to enhance the internal data structures of Dymola. In May 2000, we could simulate an entire vehicle, (Bowles, et.al., 2001).

Sven Erik Mattsson invited Hubertus Tummescheit in 1998 to make a Ph.D. thesis at the Automatic Control department, Lund. Hubertus and Jonas Eborn developed a comprehensive Modelica library for thermo-dynamical systems which broadened the applicability of Modelica. Hubertus organized the first Modelica Workshop in Lund in 2000 with 80 participants. Gerhard Schmitz and his group in Hamburg utilized this library for building a library for air-conditioning modeling. DaimlerChrysler, BMW, Volkswagen and Audi wanted to standardize on a modeling format to allow them to exchange models with their suppliers. During the autumn 2003 there was intense work including benchmarking various tools. However, we needed to form an organization that could supply and support these customers. Hubertus and Jonas then worked at UTRC, USA. I called Hubertus in December 2003 and asked him to come back to Lund and help organizing the effort. He was back before Christmas. This led to founding of Modelon AB by Hubertus Tummescheit, Jonas Eborn, Magnus Gäfvert, Johan Andreasson and me in 2004.

Peter Fritzson made a formal specification of Modelica in 1998 which later led to the open source implementation OpenModelica, (Fritzson, 2002).

Hans Olsson took the responsibility in 2000 of being the editor of the Modelica specification, i.e. to organize the process of resolving all change requests.

Peter Fritzson wrote the book **Principles of Object Oriented Modeling and Simulation with Modelica 2.1** which was published in 2003, (Fritzson, 2003).

## 5 Dassault Systèmes

Francois Bichet, Dassault Systèmes called in November 2005 and asked if he and Dominique Florack, VP of R&D could visit Dynasim. I made a presentation of the company and the technology and got invited to Paris for presentations to a larger group at Dassault Systèmes and for further discussions. I then got the question if Dynasim was for sale. I considered such a deal as a perfect marriage since I believe systems engineering, modeling and

simulation must be married to 3D and CAD. An intense period of meetings and discussions led to a letter of intent signed before Christmas 2005. The deal was closed on April 2, 2006 and announced at the Dassault Systèmes DevCon Conference in June, 2006.

The first years of joining such a big company were somewhat turbulent for us. Dymola and Modelica represented a new world for some of the managers at Dassault Systèmes HQ and the Dynasim team was too remote from HQ. The situation was corrected in the beginning of 2010 with new manager for CATIA Systems, I was appointed as Chief Technology Officer for Systems reporting to Philippe Laufer, now CEO of CATIA. Dan Henriksson became manager of the Dassault Systèmes Lund team and Martin Malmheden relocated to Dassault Systèmes in Paris to help in the technology transfer. This enabled me to focus on technical aspects and in particular, together with Sven Erik Mattsson, to work on new features such as the synchronous extensions to Modelica and multi core simulation.

## 6 Modelica Community

The Modelica community is now growing rapidly. There are 9 tools having at least partial support for the Modelica language, see [www.Modelica.org/tools](http://www.Modelica.org/tools). There are more than 1300 models in the Modelica Standard Library. A Modelica compliance test suit with more than 1000 tricky models is available for checking tool compliance. It is planned that a web page with compliance status will be available.

We are now organizing the 10<sup>th</sup> Modelica Conference. More than 22 Ph.D. theses, 800 papers and articles have been published related to Modelica.

Several large EU projects have been activated: RealSim, EuroSysLib, Modelisar, OpenProd, and Modrio with more than 100 man-years of effort.

The Modelica Association is also growing, now having more than 100 individual members and more than 10 industrial members. People from the synchronous language community have joined after Philippe Laufer suggested I should attend a meeting with them in Paris in October 2010. The synchronous Modelica features were developed with the help of Albert Benveniste, Marc Pouzet, Francois Dupont and others, (Elmqvist, et.al., 2012).

The scope of Modelica Association is also growing to also incorporate the FMI development.

## 7 Modelica Future

This section discusses a few possible directions for the future of Modelica. In most cases it is a matter of both tool support and various extensions to Modelica.

As mentioned before, systems engineering also need to take **3D aspects** into account, such as mass and inertia calculated from actual shape and materials, collision handling, etc. It is also important to have consistent parameterization of the behavior model and the shapes. It should be possible to initially use a simplified model for concept studies and gradually replace submodels when CAD parts or a kinematic model become available as illustrated below:



Modelica gives the possibility to compose more and more detailed models since model components can be reused. This means that simulation needs to be faster. One possibility is then to use multi-core technology. Recent advances with more than 1000 cores (www.Kalray.com) show the potential. The problem is then how to utilize this enormous processing power in a user friendly way. Partitioning needs to be made automatically. Modelica gives good possibility to automatically partition the model equation execution into separate threads since it is a declarative language based on equations. Early developments are described in (Aronsson, et.al. 2002). A new paper (Elmqvist, Mattsson, 2014) describes a method to **automatically parallelize model equations** implemented in Dymola. A speed-up of 3.4 times has been achieved using 4 cores/8 threads.

Modelica models and simulations is one part of systems engineering. It is important to **enable various tool chains** including sensitivity analysis, optimization, Monte Carlo simulations, linear analysis, code generation, etc. It should be possible to define a process flow that can be reused if some early requirements or designs need to be changed. It is important with standardized interfaces. A subgroup of Modelica Association is working on defining an XML format for various levels of Modelica representation, source, flattened, solved, etc. This will enable various backend tools such as fault tree analysis and code generation. Another subgroup is working on how to store meta information in a standard-

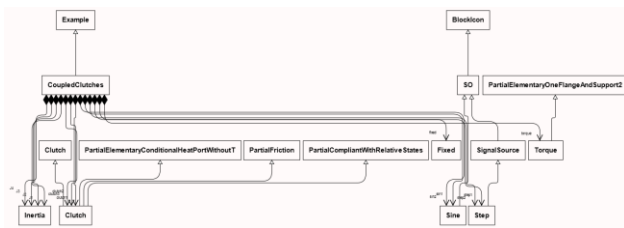
ized way, for example, for optimization (Zimmer, et.al., 2014).

A subtask within the MODRIO project is to enable **modeling of multi-mode and stochastic systems**, for example, to model failing components. A generalization of synchronous state machines to continuous time and that the number of states of a system can change at events is proposed, (Elmqvist, et.al., 2014).

Modelica now has quite limited data structures, i.e. only arrays and records are allowed. It is not possible to define recursive data structures or maps. Such **advanced data structures** are, for example, needed for advanced controllers, scripting, and meta information for work flows. MetaModelica has successfully incorporated such features already 2005 (Fritzson, 2005).

We in the Modelica Association also want to define more features of Modelica using a **Modelica Kernel language**. This would enable, for example, defining connection semantics, state machine semantics, etc. in the Modelica Kernel language. A prerequisite for this approach is that more advanced data structures are available in the Modelica Kernel language.

Most Modelica tools allow several views of a model or library such as Diagram, Icon, Modelica Text, Documentation, Package tree view. However, **SysML offers several additional useful views** such as inheritance tree, component hierarchy, etc. Such views can be automatically generated from Modelica. For example a Block Definition Diagram (bdd) of SysML for the CoupledClutches example might look as shown below using a Layered Digraph layout algorithm.



The question about **standardization** of Modelica as ANSI or IEEE standard is frequently raised. The members of Modelica Association want to wait because we feel there are some unifications and enhancements that need to be made first, such as simplification and extension of generics (redeclare) and Modelica Kernel language.

## 8 Conclusions

The Modelica effort is a truly multi-disciplinary challenge involving compiler technology, graph theory, computer algebra, numerical mathematics, etc. Furthermore, Modelica library development requires skills from many engineering domains such as electronics, mechanics, and thermo-dynamics. The beauty of Modelica is that it enables them all to express themselves in the same language.

An important driving force behind Modelica is to enable storing engineering knowledge in a form that can be reused. This enables more realistic simulations of the complex systems now being designed. It gives the engineers a more interesting work place since tedious manual model development is not needed. This also has the effect that many manual errors are avoided and the simulation results can be better trusted.

Another driving force has been and is the joy of being part of a community with the desire to change the design methodology to become model based. Many hard problems have had to be solved which means that there has been many exciting moments of success.

It seems fair to state that Modelica has already been proven as a means for storing engineering knowhow in a reusable form. The equation orientation means that equations from engineering books can often be reused directly. There is no need to manually transform the equations since the tools do that for you. It is a neutral form which already is supported by many tools. So a good stable platform has been established.

## 9 Acknowledgements

I am thankful to all persons mentioned in this paper and to many others that enabled this saga to take place.

In particular, I want to thank my thesis advisor and mentor Prof. Karl Johan Åström for guidance and for believing in me that this direction would be possible and useful. He wrote in a certificate about me: "He walks his own ways". It took several years until I understood how positive that was.

Thanks to Sture Lindahl for writing such a well-structured and comprehensive report about power systems modeling which could be formalized.

Francois Cellier was the first user to really understand the principles and power of Dymola, introducing Dymola in his courses in Tucson and in his book. Thank you, Francois.



I want to thank Georg Grübel for helping financing the early developments and for introducing Dymola in Japan.

Without the devoted work by Martin Otter, Modelica would not have succeeded. Thank you, Martin, for your dedication, hard work and fantastic collaboration during 22 years.

Yutaka Hirano and Akira Ohata understood the industrial significance of Dymola but also understood that in order to succeed, financial support was needed which I am very grateful for.

Francois Bichet followed the Modelica evolution for a long time before calling me. Thank you, Francois, for promoting Modelica and Dymola internally in Dassault Systèmes.

I want to thank my manager Philippe Laufer for believing in me and letting me use my own judgment for advancing the Modelica, Dymola and FMI technologies.

Dag Brück, Sven Erik Mattsson and Hans Olsson were absolutely essential for our success. Thank you for the creative, innovative and fun collaboration. I also want to thank all the other past and present members of the Dynasim/Dassault Systèmes Lund team for a great collaboration.

And most importantly, thanks to my wife Iréne Lind Elmqvist who has supported and encouraged me for more than 40 years and for coping with the uncertainties of a start-up company. Thank you, Iréne, for also taking me away from virtual reality to reality in places such as Namibia, South Africa, Tanzania, Botswana, Antarctica, Galapagos, New Zealand, Cook Islands; finding tigers in India and skiing in Åre.

## References

(In chronological order.)

- Elmqvist H. (1975): **SIMNON— An interactive simulation program for nonlinear systems — User's manual.** *Technical Report TFRT-7502. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.*
- Lindahl S. (1976): **A Non-linear Drum-Boiler Turbine Model.** *Technical Report ISRN LUTFD2/TFRT--3132--SE, Department of Automatic Control, Lund University, Sweden, March 1976.*
- Elmqvist H. (1977a): **SIMNON - An interactive simulation program for nonlinear systems.** *Proc. Simulation '77, Montreux, Switzerland.*
- Elmqvist H. (1977b): **A New Model Language for Continuous Systems.** *LUTFD2/(TFRT-7132), Department of Automatic Control, Lund Institute of Technology, Sweden.*
- Elmqvist, H. (1978): **A Structured Model Language for Large Continuous Systems.** *PhD Thesis ISRN LUTFD2/TFRT--1015--SE, Department of Automatic Control, Lund University, Sweden, May 1978.*
- Elmqvist, H. (1979a): **DYMOLA - A Structured Model Language for Large Continuous Systems.** *Proc. Summer Computer Simulation Conference, Toronto, Canada.*
- Elmqvist, H. (1979b): **Manipulation of Continuous Models Based on Equations to Assignment Statements.** *Proc. Simulation of Systems '79, IMACS Congress 1979, Sorrento. North Holland Publ. Comp.*
- Hennessy, J.L., Elmqvist, H. (1982): **The Design and Implementation of Parametric Types in Pascal.** *Software: Practice and Experience 12:169-184, 1982.*
- Elmqvist H. (1985): **LICS - Language for Implementation of Control Systems.** *Technical Report ISRN LUTFD2/TFRT--3179--SE, Department of Automatic Control, Lund University, Sweden, December 1985.*
- Pantelides C. (1988): **The Consistent Initialization of Differential-Algebraic Systems,** *SIAM J. Sci. Stat. Comput., 9(2), pp. 213-231.*
- Cellier F. E. (1991): **Continuous System Modeling.** *Springer-Verlag, New York, USA.*
- Elmqvist H. (1991a): **Cooperating Distributed Control objects.** *IFAC Symposium on Distributed Intelligence Systems, August 13-15, 1991, Arlington, Virginia, USA.*
- Elmqvist H. (1991b): **A Uniform Architecture For Distributed Automation.** *ISA/91 International Conference & Exhibition, Anaheim, USA, October 28-31, Instrument Society of America.*
- Elmqvist H. (1992): **An Object and Data-Flow based Visual Language for Process Control.** *ISA/92-Canada Conference & Exhibit, Toronto, Canada, Instrument Society of America.*
- Cellier, F.E., and H. Elmqvist (1992), **The Need for Automated Formula Manipulation in Object-Oriented Continuous-System Modeling,** *Proc. CACSD'92, IEEE Computer-Aided Control System Design Conference, Napa, CA, pp.1-8.*
- Mattsson S.E. and G. Söderlind (1992): **A New Technique for Solving High-index Differential-algebraic Equations.** *Proc. CACSD'92, IEEE*

- Computer-Aided Control System Design Conference*, Napa, CA, pp. 218-224.
- Cellier, F.E., and H. Elmqvist (1993), **Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling**, *IEEE Control Systems*, **13**(2), pp.28-38.
- Elmqvist, H., F.E. Cellier, and M. Otter (1993), **Object-Oriented Modeling of Hybrid Systems**, *Proc. ESS'93, SCS European Simulation Symposium*, Delft, The Netherlands, pp.xxxi-xli.
- Mattsson, S.E. and G. Söderlind (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. *SIAM Journal of Scientific and Statistical Computing*, Vol. 14 pp. 677-692, 1993.
- Otter, M. (1994): **Object-Oriented Modeling of Drive Trains with Friction, Clutches and Brakes**, *Proceedings of the European Simulation Multiconference, ESM'94, Barcelona, Spain, June 1-3, 1994*, pp. 335-339. SCS, The Society for Computer Simulation.
- Elmqvist, H., and M. Otter (1994): **Methods for Tearing Systems of Equations in Object Oriented Modeling**, *Proc. ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1-3, 1994*, pp. 326-332.
- Andersson M. (1994): **Object-Oriented Modeling and Simulation of Hybrid Systems**. *PhD thesis ISRN LUTFD2/TFRT--1043--SE*, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.
- Elmqvist H., M. Otter, and F.E. Cellier (1995), **In-line Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems**, *Proc. ESM'95, SCS European Simulation MultiConference*, Prague, Czech Republic, pp.xxiii-xxxiv.
- Otter, M., H. Elmqvist, and F.E. Cellier (1996), **Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola**, *J. Nonlinear Dynamics*, **9**(1), pp.91-112.
- Åström K.J., Elmqvist H., Mattsson S.E.: **Evolution of Continuous-Time Modeling and Simulation**, *The 12th European Simulation Multiconference, ESM'98, June 16-19, 1998, Manchester, UK*.
- Tiller M. (2001): **Introduction to Physical Modeling with Modelica**, *The Springer International Series in Engineering and Computer Science*.
- Bowles P., M. Tiller, H. Elmqvist, D. Brüch, S.E. Mattsson, A. Möller, H. Olsson, M. Otter (2001): **Feasibility of Detailed Vehicle Modeling**, *SAE 2001 World Congress*.
- Fritzson P., P. Aronsson, P. Bunus, V. Engelson, H. Johansson, A. Karström, and L. Saldamli. (2002): **The Open Source Modelica Project**. In *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, Mar. 18–19, 2002.
- Aronsson P., and P. Fritzson (2002): **Multiprocessor Scheduling of Simulation Code from Modelica Models**. In *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, Mar. 18–19, 2002.
- Fritzson, P. (2003): **Principles of Object Oriented Modeling and Simulation with Modelica 2.1**, 940 pages, ISBN 0-471-471631, Wiley-IEEE Press. Sept. 2003.
- Fritzson P., A. Pop, and P. Aronsson (2005): **Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica**. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany, March 7-8, 2005.
- Elmqvist H., Otter M., and Mattsson S.E. (2012): **Fundamentals of Synchronous Control in Modelica**. *Proceedings of 9th International Modelica Conference, Munich, Germany, September 3-5*.
- Zimmer D., Otter M., Elmqvist H., and Kurzbach G. (2014): **Custom Annotations: Handling Meta-Information in Modelica**. *Proceedings of 10th International Modelica Conference, Lund, Sweden, March 10-12*.
- Elmqvist H., and Mattsson S.E. (2014): **Parallel Model Execution on Many Cores**. *Proceedings of 10th International Modelica Conference, Lund, Sweden, March 10-12*.
- Elmqvist H., Otter M., and Mattsson S.E. (2014): **Modelica extensions for Multi-Mode DAE Systems**. *Proceedings of 10th International Modelica Conference, Lund, Sweden, March 10-12*.

# The Functional Mockup Interface - seen from an industrial perspective

Christian Bertsch    Elmar Ahle    Ulrich Schulmeister

Robert Bosch GmbH, Corporate Sector Research and Advance Engineering,

Robert-Bosch-Strasse 2, 71701 Schwieberdingen, Germany

[christian.bertsch@de.bosch.com](mailto:christian.bertsch@de.bosch.com)

[elmar.ahle@de.bosch.com](mailto:elmar.ahle@de.bosch.com)

[ulrich.schulmeister@de.bosch.com](mailto:ulrich.schulmeister@de.bosch.com)

## Abstract

The demand for model exchange between development partners will grow during the next years. The Functional Mockup Interface (FMI) is a well received tool independent approach for model exchange. The Original Equipment Manufacturers (OEM) have committed themselves to support FMI as exchange format for simulation models. Therefore, the FMI is a promising candidate to become the industry standard for model exchange and cross-company collaboration. In this paper, the FMI standard is evaluated from an industrial perspective.

*Keywords: FMI, industrial perspective, problem classification*

## 1 Introduction

There is a strong trend of virtualization in engineering, where simulation replaces real testing in order to develop faster, cheaper and with more system understanding. Therefore, system simulation is used in different phases of product development and gains importance within the entire product development cycle. The design of future products is done on the basis of simulation models.

### 1.1 Modeling and Simulation at Bosch

However, the simulation landscape in the field of system and component design in industry is very heterogeneous. At Bosch, there are more than 100 simulation tools with incompatible model representations in use, some of them are preferred tools for different physical or engineering domains. The picture becomes even more complex when looking at external partners such as Original Equipment Manufacturers (OEM) with a different set of preferred simulation tools. Today, there exist only proprietary ex-

change formats that are limited in functionality and only applicable to a limited number of tool combinations.

### 1.2 Approaches for modeling of complex systems and model exchange

There are two complementary approaches for modeling complex systems:

*White box modeling:* Modeling the entire system with one modeling language. This requires a modeling language that is suitable for different physical domains. This approach offers the possibility for deep system understanding by equation-based, object-oriented modeling and symbolic manipulation. MODELICA is an example for that approach.

*Black box model exchange:* Defining an interface for model exchange for standardized, tool-independent exchange format for simulation models is a complementary approach. It offers a way to cope with a heterogeneous simulation tool environment and allows using specialized tools for different physical domains. This approach offers also the possibility for know-how protection for model exchange in distributed collaborative system engineering. The Functional Mockup Interface (FMI) is a promising candidate to become the industry standard for this kind of model exchange and cross-company collaboration.

In the following, the black box model exchange with FMI is evaluated from an industrial perspective. Requirements on such an interface for industrial applications are

- *standardization* of the model interface (in order to be tool-independent),
- *availability* of a significant number of supporting tools,
- *easy-of-use* of the interface for simulation engineers,
- *adoption* of the standard in the specific industry domain,

- *accompanying documentation* and a reference process for the exchange of such black box models, and
- *maturity* of such an interface, i.e.,
  - no errors when importing FMUs from other sources and
  - reliable simulation results when using such an interface.

### 1.3 The Functional Mockup Interface for black box model exchange

The Functional Mockup Interface [1-4] has been developed to meet the requirements listed above. The requirement *standardization* and tool-independence is fulfilled by design. There is growing support for FMI by simulation tools (i.e., more than 40 listed on the FMI website), so the *availability* is given (although some often used simulation tools are still missing). The *easy-of-use* to some extent is also by design (e.g., XML-file for model information) and is realized quite well by the majority of exporting and importing simulation tools. The Global Automotive Advisory Group (GAAG) has committed itself to support FMI as exchange format for simulation models [5]. Thus, the broad *adoption* is at least planned. The *accompanying documentation* is addressed in the ProSTEP Smart Systems Engineering Project [6]. In the following the paper will focus on the *maturity* of the FMI standard or the maturity of the implementation respectively.

## 2 The Maturity of FMI

### 2.1 Maturity of FMI after the end of the MODELISAR project

As a result of the MODELISAR project, the FMI standard 1.0 was published and it was implemented by a growing number of modeling and simulation tools. In the MODELISAR project, requirements for FMI were derived from the beginning and tested for industrial applications. The performance of the FMI approach was demonstrated in 24 industrial applications [4]. Additional successful applications have been reported at the 8<sup>th</sup> and 9<sup>th</sup> MODELICA Conference, see e.g. [7-10]. However, these examples were realized in some fixed combinations of FMU-exporting and importing tools.

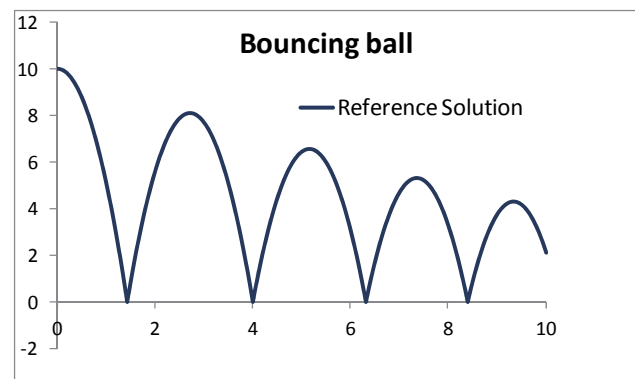
In 2012, an internal benchmark at Bosch with three exporting tools (12 test FMUs exported) and five importing tools showed quite different results. The test examples range from a “model” containing a sine generator only, a bouncing ball, a spring-damper

system, an RC circuit to a thermal network. While some combinations worked quite well, other combinations did not work at all. Typical problems at that time were

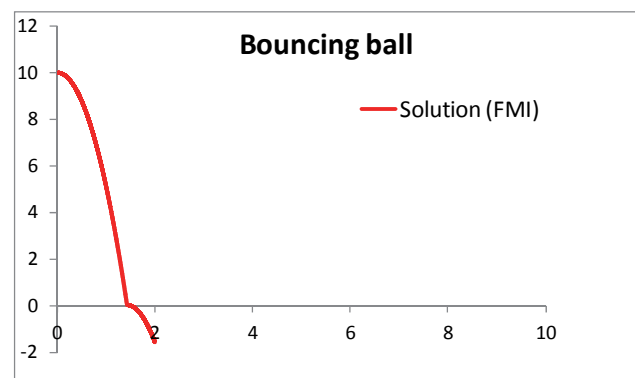
- formal errors in the XML-file,
- errors during initialization,
- memory leakage, and
- different simulation results in different importing tools.

Test results for the bouncing ball example are as depicted in Figure 1. The model was created in tool 1. Then, it was exported as a Functional Mockup Unit (FMU) for model exchange. The import of the FMU in the same simulation tool was possible, but the simulation results were erroneous, see Figure 1. The simulation of the FMU was also not possible in four other simulation tools due to different errors, e.g.,

- failed assertions,
- two importing tools crashed during simulation, or
- unspecified error at import.



(a) Reference solution



(b) FMU simulation result in tool 1

Figure 1: Comparison of simulation results for a bouncing ball example (status March 2012)

At the same time, every tool could be listed on the FMI website in “green” and claim to support FMI as

depicted in Figure 2. The number of these tools increased very rapidly, but did not reflect the experience of the internal benchmark.

Tools supporting FMI	Model Exchange		Co-Simulation		Notes
	Export	Import	Slave	Master	
AMEsim	Available	Available	Planned		Modelica environment from LMS-Imagine
ASM	Planned	Planned	Planned		AUTOSAR Builder from Dassault Systèmes
Atego Ace	Available	Available	Available		Co-simulation environment with AUTOSAR and HL support
Building Controls Virtual Test Bed	Available	Available	Available		Software environment, based on Ptolemy II, for co-simulation of, and data exchange with, building energy and control systems
CATIA V8R2013	Available	Available	Available		Environment for Product Design and Innovation, including systems engineering tools based on Modelica, by Dassault Systèmes
ControlBld	Available	Available	Available		Environment for IEC 61131-3 control applications from Dassault Systèmes
CoSim4e	Available	Available	Available		Co-simulation Environment from ChasTek
Cybernetica CENT	Available	Available	Planned		Industrial product for nonlinear Model Predictive Control (MPC) from Cybernetica
Cybernetica ModelFit	Available	Available	Available		Software for model verification, state and parameter estimation, using logged process data. By Cybernetica.
DO4plus	Planned	Planned	Planned		Fluid power simulation software from FLUIDON
Dynola 2013	Available	Available	Available		Modelica environment from Dassault Systèmes
EnergyPlus	Available	Available	Planned		Whole building energy simulation program
FMI Library (from Modelon)	Available	Available	Available		Open source (BSD) C library for integration of FMI technology in custom applications by Modelon.
FMI add-on for NI VeriStand	Available	Available	Available		Manages simulations with FMI for co-simulation V1.0, available from DCFWare
FMI SDK	Available	Available	Available		FMI Software Development Kit from QTronic.
FMI Trust Centre	Available	Available	Available		Cryptographic protection and signatures of models including their safe PLM storage, secure authentication and authorization for protected co-simulation via Modeling and Co-Simulation environment by Modelon
IPG CarMaker JFMI	Planned	Planned	Planned		
JFMI	Available	Available	Available		Java Wrapper for the Functional Mock-up Interface, based on FMI SDK
JModelica.org	Available	Available	Planned		Open source Modelica environment from Modelon
MATLAB (via FMI Toolbox)	Available	Available	Available		via FMI Toolbox from Modelon
MWorks 2.5	Available	Planned	Planned		Modelica environment from Suzhou Tongyuan

Figure 2: Beginning of list of tools supporting FMI (source: <https://www.fmi-standard.org>, March 2012)

## 2.2 A classification of possible problems in FMI-based simulation

The challenges encountered in the benchmark are classified and ordered with increasing maturity of the FMI standard:

- I. *Inconsistencies of the standard*
  - Source of problem: Standard
  - Examples: Ambiguities regarding naming conventions and structure of ZIP-file, XML model-description file
- II. *Formal errors in the FMU-File*
  - Source of problem: Exporting tool
  - Examples: Errors in XML and internal directory structure of ZIP-file
- III. *Formal interaction of importing tool with FMU*
  - Source of problem: Exporting or importing tool
  - Examples: Errors in calling functions; memory leakage

## IV. Simulation of one FMU (with solver of importing tool in case of model exchange)

- Source of problem: Exporting or importing tool
- Examples: Incorrect initialization; incorrect simulation result; incorrect event handling

## V. Simulation of multiple connected FMUs with solver for model exchange

- Source of problem: Solver or master algorithm of exporting/importing tool, overall system divided into several FMUs at unsuitable location
- Examples: Incorrect treatment of algebraic loops; numerical problems with FMUs with different time constants (stiffness)

These problem categories are summarized and ordered in Figure 3. The problem classes correspond to different stages in the simulation process. If an error occurs in one problem class, typically the simulation fails or the results are not reliable.

At the beginning of 2012, the FMI standard 1.0 was quite mature so that problem class I was not a big issue. But there occurred several problems corresponding to problem classes II to IV for exchanging even single FMUs between different tools and simulating them. Only the problem classes I to IV are FMI-specific; problem class V corresponds to typical simulation challenges of modular simulation with causal interfaces or co-simulation. Some of the problems in FMI-based simulation might be observed for the first time, because models have neither be exchanged nor been coupled in the past.

## 3 Implemented measures to improve maturity

The maturity issues of FMI-based simulation were addressed to the FMI community at the MODELICA/FMI meetings beginning early 2012. The discussions and input from other companies and users resulted in the call for quotation of an FMU

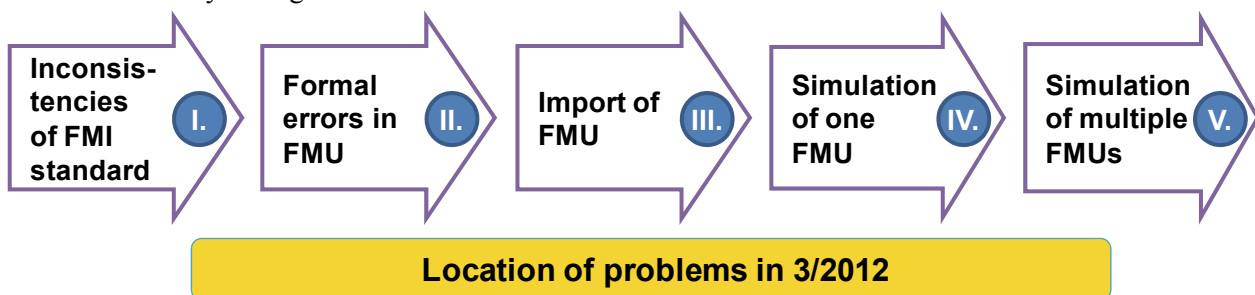


Figure 3: Classification of problems in FMI-based simulation

Compliance Checker and later of FMI Cross Checking rules. Bosch’s application for membership in the FMI Steering Committee was accepted in January 2013. Since then Bosch is actively contributing at FMI design and steering committee meetings.

### 3.1 FMU Compliance Checker

The FMU Compliance Checker [11] is a software tool that was initiated by the MODELICA Association by a call in 04/2012 and was implemented by the company MODELON AB. It addresses problem classes II, III, and also partly IV (the tool contains only a very simple solver). In the first release only one FMU could be called and no inputs to the FMU could be provided. With a later release inputs to FMUs can be provided as CSV-files. The FMU Compliance Checker is a very valuable tool to check for formal errors of the FMU (see Figure 4).

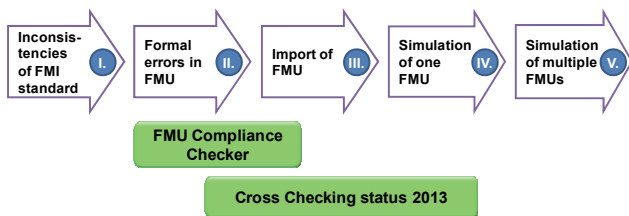


Figure 4: Focus of FMU Compliance Checker and FMI Cross Checking

### 3.2 FMI Cross Checking

The FMI Cross Check Rules [12] were approved in February 2013 and focus on testing the quality of implementations of exporting and importing tools. It is defined how many FMUs an exporting tool must publish and for how many of these FMUs an importing tool must publish simulation results in order to get listed as “available” on the FMI-tools website [3]. As at the moment only single FMUs are tested, FMI Cross Checking at the moment focuses on problem class III and IV as depicted in Figure 4. As a side effect it will also show problems of class I and II.

Figure 5 shows a screenshot of the FMI Tools website [3] and an entry in green refers to an exporting or importing tool that has successfully passed FMI Cross Checking. An entry in orange shows that the corresponding FMI export or import is claimed to be available but is not tested according to the FMI Cross Check Rules.

**FMI Support in Tools**

**Compatibility Table**

Generated on 2013-11-07 09:52 UTC

Legend FMI Support: ▶ Planned, ▶ Not available yet, ▶ No CrossCheck results submitted, ▶ Available, ▶ Passed CrossCheck, press for results

More information about the generation of the CrossCheck results can be found in the [Rules document](#) and the [Implementation notes](#).

The following modeling and simulation environments support or plan to support FMI 1.0 (alphabetical list):

Tools supporting FMI	Model-Exchange		Co-Simulation		Notes
	Export	Import	Slave	Master	
Adams	Planned	Available	Available	Available	High end multibody dynamics simulation software from MSC Software
AMESim	Available	Available	Available	Planned	Modica environment from LIS-Imagine
ANSYS Smplicer	Available	Planned	Planned	Planned	ANSYS Smplicer is a multi-domain, multi-technology simulation program from ANSYS.
ASim - AUTOSAR Simulation	Available	Available	Available	Available	AUTOSAR product from Dassault Systèmes
Atago Ace	Available	Available	Available	Available	Co-simulation environment with AUTOSAR and HIL support
@Source	Available	Available	Available	Available	Simulink via @Source
Building Controls Virtual Test Bed	Available	Available	Available	Available	BCVTB is a Software environment, based on Proterity 1, for co-simulation of, and data exchange with, building energy and control systems.
CarMaker	Available	Available	Available	Available	CarMaker is an open test- and integration-platform for ML, SL, and HL.
CATA	Available	Available	Available	Available	Environment for Product Design and Innovation, including systems engineering tools based on Modelica, by Dassault Systèmes
ControlBuild	Available	Available	Available	Available	Environment for EC 61131-3 control applications from Dassault Systèmes
Coslate	Available	Available	Available	Available	Co-simulation Environment from ChaeTek
Cybernetica CENIT	Available	Available	Available	Planned	Industrial product for nonlinear Model Predictive Control (MPC) from Cybernetica.
Cybernetica ModelFi	Available	Available	Available	Available	Software for model verification, state and parameter estimation, using logged process data. By Cybernetica.
DSiPlus	Planned	Planned	Planned	Planned	Fluid power simulation software from FLUIDON
Dynola	Available	Available	Available	Available	Modica environment from Dassault Systèmes. ModelExchange also available for Simulink using Simulink Coder.
EnergyPlus	Available	Available	Available	Available	Whole building energy simulation program
FMI Add-in for Excel	Available	Available	Available	Available	FMI Add-in for Microsoft Excel by Modelon. Offers support for batch simulation of FMUs.
FMI add-on for NI VeriStand	Available	Available	Available	Available	NI VeriStand supports FMI through the use of the FMI add-on for NI VeriStand from Dofware
FMI Toolkit for Simulink	Available	Available	Available	Available	Import of FMI Co-Simulation models into Simulink - provided by Clayer.
FMI Library	Available	Available	Available	Available	Open source (BSD) C library for integration of FMI technology in custom applications by Modelon.

Figure 5: Beginning of list of tools supporting FMI (source <https://www.fmi-standard.org>, status 11/08/2013)

At first glance, it is disappointing that there are so many “orange” entries of exporting tools that have not yet provided test FMUs and of importing tools that have not yet published simulation results. On the other hand this makes it transparent that there are combinations of FMU-exporting and importing tools that are tested and other that are not. This is exactly what was intended with the FMI Cross Checking rules and helps the user to select tools that he can rely on.

**CrossCheck Results for FMI\_1.0**

Variant: ModelExchange

Platform: Linux32

Generated on 2013-11-07 09:52 UTC

Legend FMI Support: ▶ 3 FMI's imported successfully, ▶ 1 FMI rejected, ▶ 2 FMI's failed test

FMI 1.0	Exporters	CATA	ControlBuild	Dynola	FMI Toolbox for MATLAB	JModelica.org	LMS Virtual Lab Motion	MapleSim	OPTIMICA Silver	SimulationX		
ModelExchange Win32	4 exporters	VER2013a	2013-2a	2013-FMI 1.5	1.6	1.5.1	1.10	RevH15L2	6.1	1.244	2.6.0.312_alpha12	3.5.707
Dynola	2014	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28	2013-08-28
FMI Library	2,662	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05	2013-06-05
FMI Toolbox for MATLAB	1.5-MEX	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19
FMI Toolbox for MATLAB	1.5-Simulink	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19	2013-05-19
JModelica.org	1.5.1	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20
MapleSim	1.10	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11
OPTIMICA Studio	1.244	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20	2013-05-20
PyFMI	1.5.1	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11	2013-07-11
PyFMI	1.2.1	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15	2013-05-15
Silver	2.6.0.312_alpha12	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09	2013-06-09
SimulationX	3.5.707	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20	2013-09-20

Figure 6: Results of FMI Cross Checking, (source <https://www.fmi-standard.org>, status 11/8/2013)

For the tools that participate in FMI Cross Checking, detailed results are published on the FMI website, as depicted in Figure 6. Thus, FMI Cross Checking is very valuable for industrial users, as they are supported with information, which combinations of exporting and importing tools have already

been tested successfully for some applications. FMI Cross Checking is a source to detect improvement potential of the simulation tools.

### 4 Proposed next steps

The actions that we have seen until now address mainly the problem classes II to IV. In order to address “real-life problems”, FMI Cross Checking should be extended to more tools and for multiple FMUs. The (co-)simulation techniques for importing tools should be improved and the improvements of FMI 2.0 should be implemented soon.

#### 4.1 Extension of FMI Cross Check Rules

There should be more exported test FMUs provided by much more exporting tools und much more results of importing tools be published. FMI Cross Checking should be extended to *connected FMUs*. These FMUs could come from different exporting tools.

A simple test case is shown in Figure 7: An electrical motor with a gear and load torque (plant model) with PI controller (model of the software). The plant model and the control model are created in different MODELICA based simulation tools, then (separately) exported as FMUs and re-imported in different tools. This is a simple test example for a closed loop control, where the control software FMU might come from a different source than the plant model FMU. In some cases this worked already well from the beginning (2012), in other cases bugs have been fixed.

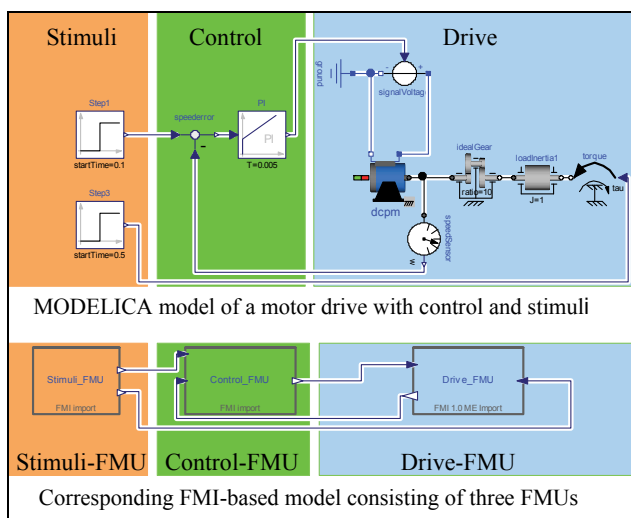


Figure 7: Example of connected FMUs exported from a MODELICA model

Further test examples of connected FMUs should be provided, e.g.,

- a stiff mechanical system consisting of various spring and damper elements exported as several FMUs, or
- a hybrid powertrain containing of different FMUs:
  - Simple map-based model of a combustion engine,
  - mechanical powertrain with gears and clutches,
  - electrical machine,
  - power electronics, and
  - control software model for the electrical machine.

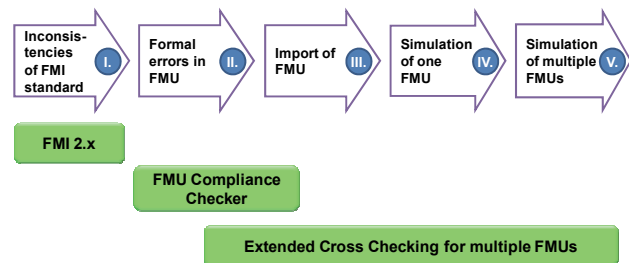


Figure 8: Focus of FMI 2.x, FMU Compliance Checker and extended FMI Cross Checking

#### 4.2 Extension of test cases to multiple connected FMUs

In order to test multiple connected FMUs in different tools easily a standardized file format for the definition of the set-up is proposed. This file should contain the following information:

- Simulation settings (e.g., simulation duration),
- list of the included FMUs,
- parameters of the included FMUs, and
- list of connections (i.e., which output of an FMU is connected to which input of another FMU).

Such a file should be preferably implemented as an XML file. All stimuli can also be defined in FMUs or provided as CSV file, as it is already possible for single FMUs within cross-checking. All files for a particular test should be either provided in a ZIP-file or a directory. With the help of such a standardized file describing the connections of multiple FMUs, the testing of connected FMUs can be done automatically in batch runs.

### 4.3 Improvement of (co-)simulation techniques of importing tools

New co-simulation techniques and master algorithms are proposed in the scientific community, see e.g. [13-15]. These new methods will improve co-simulation techniques. Also for model exchange new solver techniques should be developed that can handle multi-rate systems divided in several FMUs. The (co-)simulation capabilities of the importing tools for multiple FMUs can be compared by extended FMI Cross Checking.

### 4.4 Improvements of the FMI standard with FMI 2.0

Problem class I is addressed by bug fixes of the FMI standard. Maturity issues of the FMI standard 1.0 are not considered problematic, but additionally to new features, there will be some minor inconsistencies eliminated. FMI Cross Checking is an integral part of the introduction phase of FMI standard 2.0. Hence, it is expected that possible problems in the standard (problem class I) and the implementations (problem classes II to IV) will be detected at an early stage. FMI standard 2.0 will be only released, when a significant number of test implementations are available.

## 5 Outlook

### 5.1 Usage of FMI at Bosch

FMI enables the exchange of models between tools where this either had not been possible in the past or only by proprietary exchange formats (resulting in some cases in additional license fees). Additionally, FMI could replace existing in-house solutions of tool couplings and co-simulation. The use of a widely accepted standard is much more effective than developing and maintaining special interfaces.

### 5.2 FMI-based collaboration with OEMs

One motivation of support FMI is the increasing demand to exchange simulation models between OEM and suppliers [10]. While FMI as a technical standard is right on track, there are other points to be addressed such as model exchange process and accompanying documentation. This is developed in the ProSTEP Smart Systems Engineering Project (SSE) [6] in cooperation with OEMs. The main question there is how collaborative simulation-based engineering based on FMI can work. FMI is the best

available approach for tool-independent model exchange and co-simulation. FMI Compliance Checking and FMI Cross Checking address technical problems that existed in the past. FMI Cross checking should be extended to more complicated examples and also with multiple FMUs. Then, FMI can become the technical basis for model-based collaborative engineering in a heterogeneous tool environment with different partners.

## References

- [1] Blochwitz, T. et al., The Functional Mockup Interface for Tool independent Exchange of Simulation Models, In: Proceedings of the 8<sup>th</sup> International Modelica Conference 2011, Dresden, Germany
- [2] Blochwitz, T. et al., The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, In: Proceedings of the 9<sup>th</sup> MODELICA Conference 2012, Munich, Germany
- [3] FMI project website, <https://www.fmi-standard.org/>
- [4] Chombard, P., Multidisciplinary modeling and simulation speeds development of automotive systems and software, ITEA2 innovation report, published online: <http://www.itea2.org/project/index/view/?project=217>
- [5] Chombard, P., Standard FMI pour l'échange et la co-simulation de modèles multidisciplinaires, LMCS Conference 2012, <http://www.acsysteme.com/fr/lmcs-2012>
- [6] ProSTEP Smart Systems Engineering (SSE) Project, <http://www.prostep.org/>
- [7] Abel, A. et.al., Functional Mockup Interface in Mechatronic Gearshift Simulation for Commercial Vehicles, In: Proceedings of the 9<sup>th</sup> MODELICA Conference 2012, Munich, Germany
- [8] Sun Y., Vogel S., Steuer H., Combining Advantages of Specialized Simulation Tools and Modelica Models using Functional Mockup Interface (FMI), In: Proceedings of the 8<sup>th</sup> MODELICA Conference 2011, Dresden, Germany
- [9] Chrisofakis E., Junghanns A., Kehrer C., Rink A., Simulation-based development of automotive control software with Modelica, In: Proceedings of the 8<sup>th</sup> MODELICA Conference 2011, Dresden, Germany



- [10] Schneider, Stefan-Alexander, Virtual System Prototyping in Automotive Industry and the role of FMI, 7<sup>th</sup> MODPROD Workshop on Model-Based Product Development 2013, Linköping, Sweden
- [11] FMU Compliance Checker, <https://www.fmi-standard.org/downloads>
- [12] FMI Cross Check Rules, [https://svn.fmi-standard.org/fmi/branches/public/CrossCheck\\_Results/](https://svn.fmi-standard.org/fmi/branches/public/CrossCheck_Results/)
- [13] Schierz, T., Arnold, M., Clauß C., Co-simulation with communication step size control in an FMI compatible master algorithm, In: Proceedings of the 9<sup>th</sup> MODELICA Conference 2012, Munich, Germany
- [14] Petridis, K., Klein, A., Beitelschmidt, M.: Asynchronous method for the coupled simulation of mechatronic systems. In: PAMM Volume 8 (2008), Nr. 1
- [15] Petridis, K.: Synchrone und asynchrone Verfahren zur gekoppelten Simulation mechatronischer Systeme, TU Dresden, Dissertation, to appear



# An FMI-Based Tool for Robust Design of Dynamical Systems

Maria Henningsson, Johan Åkesson, Hubertus Tummescheit  
Modelon AB / Modelon Inc.

## Abstract

Concepts from quality sciences, such as robust design, six-sigma, and design-of-experiments have had a large impact on product development in industry. These concepts are increasingly used in a model-based engineering context where data is gathered from simulation models rather than laboratory setups or prototypes.

This paper presents a framework to apply such ideas to analysis of dynamical systems. A set of tools that can be used for uncertainty analysis of dynamical Modelica models is presented. These tools are made available in the FMI Toolbox for MATLAB. The workflow and tools are demonstrated on a cooling loop design problem.

*Keywords:* *Design-of-experiments, Robust design, Controls, Modelica, FMI*

## 1 Introduction

Model-based engineering is a key technology for competitive product development. However, implementing, parameterizing, and validating simulation models of physical systems is time-consuming and costly. To make modeling efforts pay off, it is necessary to systematically consider tools, practices, and workflows to get the most use out of a model portfolio.

In this paper, we present a tool-chain and a methodology to efficiently integrate concepts from robust design and design-of-experiments to design and analysis of dynamical systems. Robust design is a well-established methodology that aims to design products and systems such that they are inherently robust to variations in components and operating conditions. A large amount of research has been directed towards robust design methodologies, that includes concepts such as design of experiments, quality engineering, critical parameter management, and six sigma [8, 5]. In the last years, there has been a growing interest in applying these kinds of techniques to analyze detailed simulation models, see e.g. [1, 2, 6, 7, 10, 4].

The tools presented in the paper are integrated into the FMI toolbox for MATLAB. Functional Mockup Interface (FMI) is a standard that allows importing and exporting dynamical models between different tools. First released in 2010, the standard has quickly been adopted in industry and is currently supported by a large range of tools used for physical systems modeling. FMI is a powerful standard for deploying tool-independent workflows and processes. Together with physical dynamical simulation models from Modelica, it provides a suitable platform for applying robust design concepts to analysis of dynamical systems.

The paper starts by discussing typical applications of simulation models, and providing an overview of design-of-experiments (DoE) concepts. The ideas behind the design of the new DoE tools in the FMI toolbox for MATLAB are then described. The use of the tools is then illustrated for the design of a cooling system, including both static analysis to size the components, and dynamic analysis to design a controller.

## 2 Applications of simulation models

Simulation model development represents a significant strategic investment in industries such as process, automotive, aerospace, and energy. Simulation models are, however, never an end in themselves but rather a means to answer questions in the engineering design process. Such questions may be:

- Hardware optimization: find dimensions, settings, and operating points for physical components
- Verification: check that performance meets specs in entire operating envelope
- Quality: check probabilistically that system specs are satisfied given tolerances on components
- Controls: design, analyze, and test control algorithms

Simulation models should ideally be developed such that they can provide answers to all of these questions. In the literature that address model-based engineering, it is interesting to note that there is a substantial cultural difference between research publications in the control community compared to the first three items in the list.

Robust design methodologies generally consider coarse large-scale mathematical and statistical models of complex systems. The models may be based on data from physical experiments or supplier spec sheets, and are generally steady-state [10]. The literature is heavily focused on processes, workflows, and tools.

Control engineering literature has a strong mathematical and analytical focus. Dynamics are central, and simplified mathematical model classes are analyzed with rigor. Much research is focused on mathematical analysis but there is surprisingly little written on processes and practices for control design in the product development process. It is commonly noted that there is a significant gap between much of the mathematically-oriented control research community and industrial practice.

In practice, control design is often based on linearization at a single operating point. For the design to be successful, this operating point needs to be representative of the dynamics across the operating range. It is normally difficult to know how the dynamics of the process vary with operating points and parameter uncertainty. Control design methods that go beyond linear models are often cumbersome to use and require a detailed understanding of which nonlinear effects or other process uncertainty that will be relevant. In-depth domain experience or trial-and-error is generally needed to find these dominant effects.

In practice, PID controllers based on simple experiments or linearized models are widely applied in industry, even though all real-world processes have some amount of nonlinearity or uncertainty. It is not uncommon to find controllers that perform poorly, with slow responses or oscillations at off-design operating conditions.

In this paper, we suggest how approaches from the robust design field can be applied to analysis of dynamical simulation models and control design. In order to use information on process nonlinearities and uncertainty in control design, it is imperative that tools be designed so that the relevant information can be extracted in a convenient and intuitive format.

### 3 Design-of-experiments

The term design-of-experiments (DoE) denotes methodologies to gather information from a system or process in a systematic way. Originally, it was introduced as a means to collect statistically sound experimental data sets for establishing cause-and-effects relationships.

In recent years, there has been an increasing interest in applying DoE techniques to extract data from simulation models [2, 1].

#### 3.1 Terminology and designs

In the DoE terminology, a *factor* denotes a quantity that is to be varied in the data set. An *experiment* is the procedure of testing the system with a particular choice of factor settings. A *response* is an outcome that is measured in the experiment. A *test matrix* is a list of factor setting combinations to be tested.

Selecting an appropriate test matrix is key to design-of-experiments. Figure 1 shows examples of possible test matrices in two dimensions. The first plot to the left shows a design based on one-factor-at-a-time (OFAT). Here, a nominal operating point is chosen, and off-nominal behavior is checked by varying one factor at a time. This is generally not an effective way to gather information on the system, since no interactions between the variables are considered. Nevertheless, this approach is often applied in an ad-hoc manner to test robustness.

The second plot shows a full-factorial test matrix design. This is equivalent to a multi-dimensional grid, where two or more different levels are chosen for each factor and the test matrix is then assembled from all combinations of the different levels. This method may work well in two dimensions but scales poorly with the number of factors. Other designs that are still based on fixed factor levels but that only select a subset of combinations are often preferred. Examples of such designs are fractional factorial designs, Box-Behnken designs, or central composite designs.

When DoE is applied to physical experiments, it can be advantageous to use a limited set of levels for each factor. For example, each different value of the factor could involve building a separate prototype. In simulation models, however, these constraints are generally not present and the test matrix can be constructed by spreading the test points freely in the ranges of interest. Monte-Carlo simulations spread the test points according to statistical distributions of the factors, which allows to get statistical estimates on distributions of

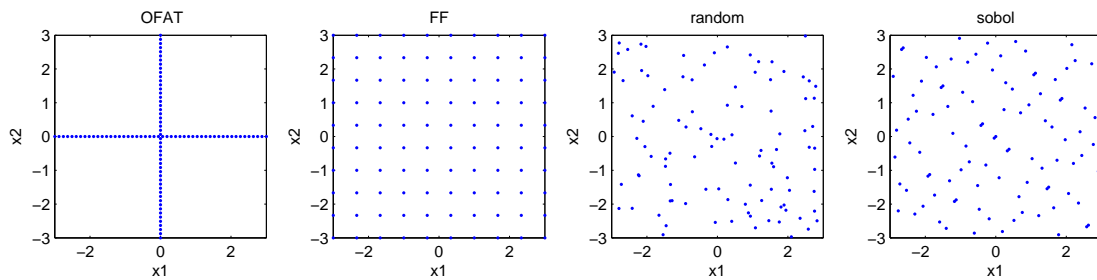


Figure 1: Example of test designs in two dimensions. OFAT = one-factor-at-a-time, FF = full factorial. Each of the four examples contain 100 test points.

the estimates. The third plot in Fig. 1 shows points that have been randomly sampled from uniform independent distributions for  $x_1$  and  $x_2$ .

To maximize the information that can be extracted from the process in a limited number of simulations, the points should fill the space spanned by the factors. Quasi-Monte Carlo designs use space-filling algorithms to spread the test points in a way that covers the space better than random sampling. The right-most plot in Fig. 1 shows a DoE design based on the Sobol space-filling algorithm. The advantages of using space-filling QMC designs instead of random sampling can be substantial in higher dimensions where corner-cases are poorly covered by random sampling [3].

### 3.2 Meta-models

The results of a DoE experiment consist of a list of pairs of factor values and response values  $\{x_i, y_i\}_{i=1..N}$  where  $N$  is the number of experiments. In some cases, it may be sufficient to verify that all  $y_i$  fulfill specifications, or to pick one  $x_i$  that give the desired outcome of the response. Often, however, we are interested in finding a factor combination  $x^*$  in a continuous set  $\mathcal{X}$  that optimizes some criteria defined by  $x$  and  $y$ . That is, it is necessary to interpolate between the points in the test matrix. To do that, some regularity on the map from  $x$  to  $y$  is assumed.

One way to find  $x^*$  is to perform sequential DoE in refined factor ranges close to the expected optimum. To limit the number of required experiments, it is generally better to use the existing data to generate a *meta-model* of the mapping from  $x$  to  $y$ . A meta-model is a simple empirical model that is obtained by fitting the data to some generic model structure, e.g. using linear regression, splines, neural networks or gaussian processes. Meta-models may also be referred to as surrogate models or emulators.

The role of meta-models is to replace the real sys-

tem with a simpler representation for analysis such as optimization or verification.

## 4 FMI tools for dynamical systems DoE

### 4.1 Modelica and FMI as a platform

Modelica has several attractive features as a platform for robust design of dynamical systems. Modelica component models normally have a sufficient level of detail to study influence of component variability, while still being simple enough to run large batches of experiments. From the nonlinear DAE models, information on the effect of both design parameters, operating conditions, and actuator settings can be investigated.

The FMI standard provides increased flexibility when model development and model analysis can be separated to different tools. The strengths of different tools and the skills engineering teams have developed using specific tools can be put to use more efficiently than if each tool needs to provide features for each potential modeling application. This facilitates cross-team use of the same model portfolio for different purposes.

MATLAB/Simulink is widely used as an environment for control design and implementation, and is thus a natural tool for analysis of model dynamics.

### 4.2 Tool requirements

The objective of a Modelica-based DoE tool is to remove the hurdles that make engineers resort to ad-hoc one-factor-at-a-time robustness testing.

The user wants to get a general sense of how sensitive the system is to parametric uncertainty. To do this, the user should be in charge of problem formulation, and provide the following input:

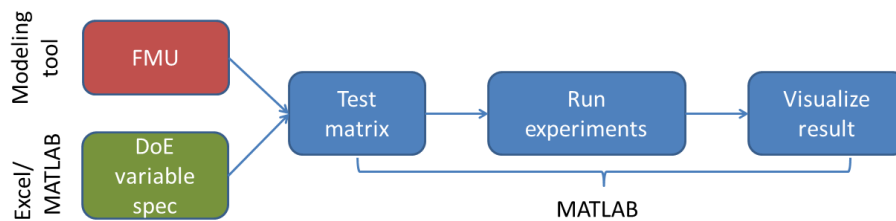


Figure 2: Workflow

- FMU model to be used
- FMU parameters, inputs, and outputs that are factors in the DoE
- Ranges or distributions for FMU parameters
- Number of experiments in the test matrix
- Type of DoE design
- Response variables to analyze or visualize

From this information, the tools should handle the back-end work:

- Construct a test matrix based on the desired DoE design and factor ranges or distributions
- Set FMU parameters
- Simulate the FMU at all points in test matrix, finding steady-state
- If the user specifies values for FMU outputs, find inputs that give these outputs
- Catching errors so that an entire batch of results is not wasted by a single test point that failed to simulate
- Linearizing the system at all points in the test matrix
- Provide convenient methods to visualize the results
- Construct suitable meta-models for analysis

### 4.3 DoE features in the FMI toolbox for MATLAB

The DoE tools in the FMI toolbox for MATLAB were introduced in release 1.6 with the purpose to make it easy and intuitive to perform DoE batch experiments on FMU models. The workflow is summarized in Fig. 2. The user specifies the names and distributions of DoE factors in either an Excel spreadsheet or a MATLAB file.

A class, `FMUDoESetup` holds the information on the FMU model to be simulated, the factor distributions, and simulation options. A batch of experiments is run by calling methods of `FMUDoESetup` that correspond to different DoE designs: full factorial, Sobol-sequence based, or Monte Carlo random simulation. There is also a possibility to use a custom test matrix. The DoE factors can be both FMU parameters, FMU inputs, or FMU outputs. If outputs are defined as factors, an equivalent number of inputs must be defined as 'free' inputs with min- max- and nominal values. These inputs are then optimized iteratively to obtain the specified outputs at each test point.

These methods return an object of the class `FMUDoEResult` that stores experiment results, including input-, output-, and state data, simulation status, and linearization at all test points. Data can be visualized by calling methods of the `FMUDoEResult` class to plot the main effects between the factors and responses, or to generate Bode diagrams or step responses for the ensemble of linearized systems at all test points.

In the future, support for meta-models may be added. An interesting application of meta-models generated from the DoE is to provide simplified subsystem models that can be used as a substitute for the full Modelica subsystem model in simulations of larger systems.

## 5 An example

This section demonstrates a detailed example on how the tools are used in the design of an engine cooling system with feedback control. The design process consists of the following steps:

- Verifying feasibility of the suggested cooling loop architecture
- Sizing the components
- Verifying the design for worst-case heat load
- Designing a robust controller for the system
- Verifying the closed-loop dynamics

Note that the example is chosen to illustrate the tools and methodology; the architecture and parameter values do not directly correspond to any existing system.

### 5.1 Model and specs

We consider a cooling loop architecture where the coolant temperature is actively regulated through an electronically controlled coolant pump. Active control of coolant temperature is achieved through modulating the coolant pump speed, as opposed to the conventional cooling system architecture where a passive thermostat valve adjusts the coolant flow to maintain safe coolant temperature. Better control of coolant temperature can help the engine management system achieve better over-all fuel economy [9].

A Dymola model of the cooling loop is shown in Fig. 3. The model is an example model provided in the Modelon Liquid Cooling Modelica library. Inputs and outputs were added to the model (inputs: heat flow from engine, pump speed, and air mass flow through radiator, outputs: pump mass flow, liquid temperature drop over radiator, temperature of liquid after the engine, and temperature of the liquid at the pump entrance).

The following specifications are set for the system

- The engine-out coolant temperature  $T_{coolant}$  must not exceed 100 °C (373.15 K).
- The system should handle a heat load of  $Q_{flow}=100$  kW
- The ambient temperature operating range is  $T_{ambient} \in [-20^{\circ}\text{C}, 45^{\circ}\text{C}]$

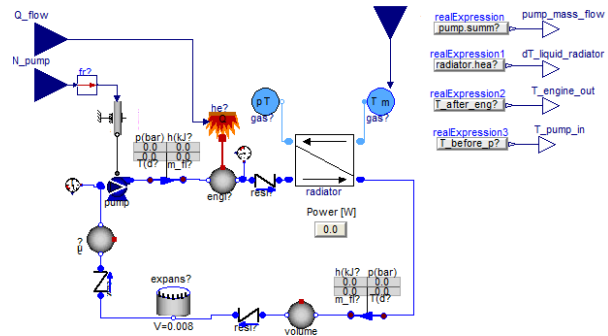


Figure 3: The cooling system model in Dymola

Three design parameters are considered: the maximum pump speed  $N_{pump}$ , the heat exchange efficiency  $\eta$  of the radiator, and the capacity of a fan that circulates air through the radiator when the ram air flow is not sufficient to provide cooling.

### 5.2 Sizing the system

The first step in the design process is to screen the design space to see if there exist some combination of parameters that meet the specifications. For this task, we look at the worst-case heat-load: maximum engine heat load and maximum ambient temperature. We let the three design parameters be factors in the DoE setup, and choose wide ranges for these factors to get a sense of feasibility of the design. The setup is summarized in Table 1.

Table 1: Experiment setup for screening the design space

variable	dist	min	max	value
$Q_{flow}$ [W]	constant			1e5
$T_{ambient}$ [K]	constant			318.15
$N_{pump}$ [rpm]	uniform	50	2000	
$mflow_{gas}$ [kg/s]	uniform	0.5	5	
efficiency	uniform	0.4	0.9	

We run a batch of 100 experiments with a Sobol QMC-design where the three factors are uniformly distributed between their maximum and minimum values in a cube. Figure 4 shows the resulting steady-state temperature plotted against each of the DoE factors. The green dots represent test points where the specification on engine-out coolant temperature is fulfilled, and the red dots represent test points that do not meet the specification. Not surprisingly, the test points that

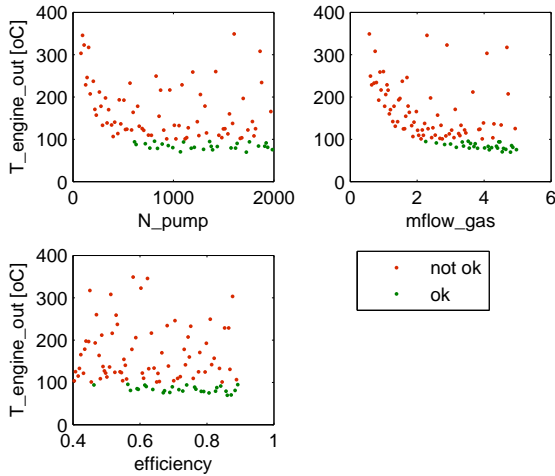


Figure 4: Result of screening design

fulfill the specs have a high pump speed, a high radiator gas flow, and a high heat exchanger efficiency.

The next step is to zoom in the design variables to a narrower range to find appropriate component specifications that meet the subsystem specification on engine-out coolant temperature. The DoE setup for this sizing task is given in Table 2. A new set of uniformly distributed test points in the new factor ranges was generated, and the effects on coolant temperature are shown in Fig. 5.

Table 2: Experiment setup for sizing the system

variable	dist	min	max	value
Q_flow [W]	constant			1e5
T_ambient [K]	constant			318.15
N_pump [rpm]	uniform	800	1200	
mflow_gas [kg/s]	uniform	2	4	
efficiency	uniform	0.6	0.8	

From Fig. 5, we can now decide on a set of component specs on the pump and radiator that would meet subsystem specification. This would normally involve a trade-off between cost and availability of components. We will here choose the design

$$\begin{aligned}
 N_{pump}^{max} &= 650 \text{ rpm} \\
 \eta &\geq 0.65 \\
 m_{flow} &\geq 3 \text{ kg/s}
 \end{aligned}
 \quad (1)$$

At worst-case conditions in terms of heat load and ambient temperature, this design gives  $T_{coolant} = 96.8^\circ\text{C}$ .

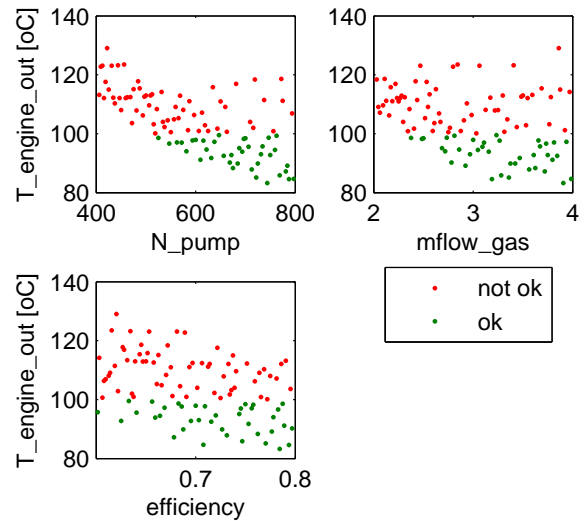


Figure 5: Result of sizing experiments

### 5.3 Dynamics

In the next step, we examine the dynamic response from pump speed command to engine-out coolant temperature. The DoE factor setup is given in Table 3. The factors were chosen to represent a range of operating conditions for the system.

Figure 6 shows the Bode diagram of the linearized systems at each point in the test matrix. It can be seen that the system is nonlinear: there is a large difference in the frequency response at different operating points.

We can now investigate the influence of the DoE factors on the dynamic response. Figure 7 shows the influence of the DoE factors on the steady-state gain of the linearized systems. The pump speed is the factor that has the largest influence on the steady-state gain.

### 5.4 Control design

The Bode diagram for the ensemble of linearizations in Fig. 6 can be used to design linear controllers that will have sufficient phase margins at all linearization points.

For nonlinear systems, there are no general guarantees that a controller that stabilizes each possible linearization will globally stabilize the closed-loop nonlinear system. In many cases, however, this is more of an academic concern. The alternative to looking at the ensemble of linearization may be to look at linearization in a single point. The ensemble of linearizations provides significantly more information.

A PI-controller with  $K = -50$  and  $T_i = 100$  was de-



Table 3: Experiment setup for evaluating the design

variable	dist	min	max	nominal
Q_flow [W]	free	0	1e5	1e4
T_ambient [K]	uniform	253.15	318.15	
N_pump [rpm]	uniform	20	650	
mflow_gas [kg/s]	uniform	3	10	
efficiency	uniform	0.65	0.8	
T_engine_out [K]	uniform	350	370	

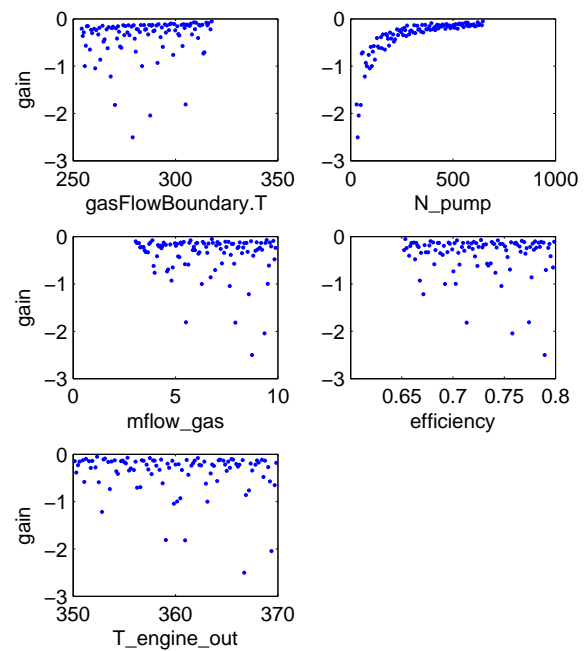
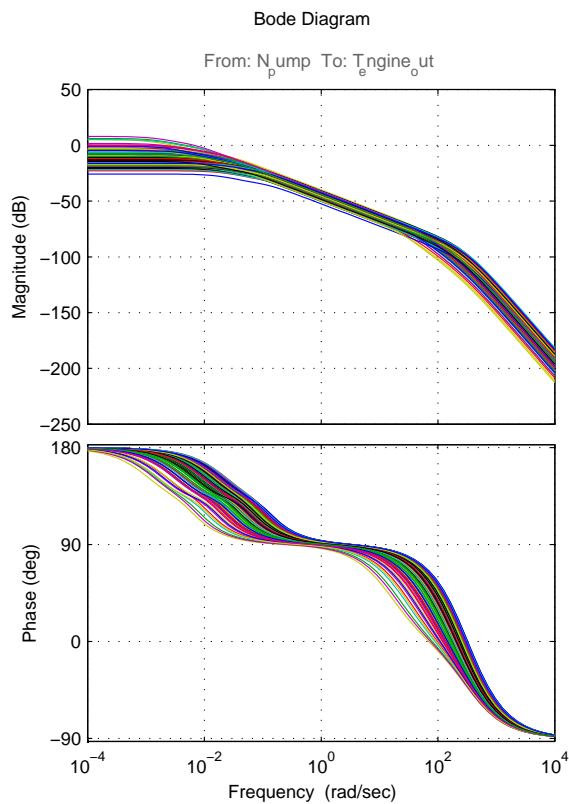


Figure 6: Ensemble Bode diagram showing the magnitude and phase for linearizations at all points in the test matrix.

Figure 7: Steady-state gain from  $N_{pump}$  to  $T_{coolant}$  of linearized system at all test points

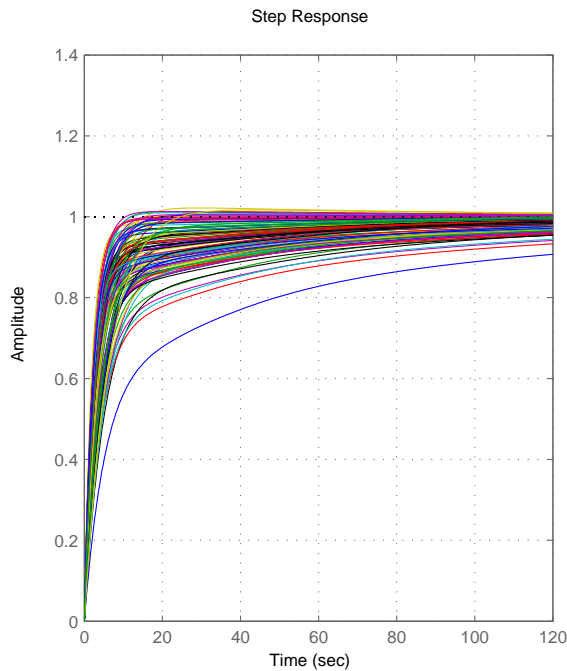


Figure 8: Step response of the closed-loop system

signed based on the ensemble Bode diagram.

Figure 8 shows the ensemble step response of the closed-loop systems corresponding to the linearizations at each point in the test matrix.

## 6 Conclusions

We have presented tools and workflows to apply robust design methods to dynamical models. The tools are available in the FMI Toolbox for MATLAB. The aim in developing these tools has been to provide methods to work with parametric uncertainty in dynamical models in a lightweight and intuitive manner.

Each of the analysis and design steps in the cooling loop example involve less than ten MATLAB commands. By taking care of back-end work, the tools allow engineers to get answers from models in a systematic fashion rather than ad-hoc testing.

It would be interesting to see an increased academic focus on design process, tools, and workflows for control design. In this context, Modelica and the FMI standard provide a powerful platform.

## References

- [1] D. W. Apley, J. Liu, and W. Chen. Understanding the effects of model uncertainty in robust design

with computer experiments. *Journal of Mechanical Design*, 128:945–957, 2006.

- [2] R. A. Bates, R. S. Kenett, D. M. Steinberg, and H. P. Wynn. Achieving robust design from computer simulations. *Quality Technology & Quantitative Management*, 3(2):161–177, 2006.
- [3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [4] D. Bouskela, A. Jardin, Z. Benjelloun-Touimi, P. Aronsson, and P. Fritzson. Modelling of uncertainties with Modelica. In *Proceedings of the 8th International Modelica Conference*, pages 673–685, 2011.
- [5] C.M. Creveling, J.L. Slutsky, and D. Antis. *Design for Six Sigma*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2003.
- [6] B. Johansson and P. Krus. Probabilistic analysis and design optimization of Modelica models. In G. Schmitz, editor, *Proceedings of the 4th International Modelica Conference*, pages 247–254, 2005.
- [7] P. N. Koch, R.-J. Yang, and L. Gu. Design for six sigma through robust optimization. *Structural and Multidisciplinary Optimization*, 26:235–248, 2004.
- [8] K. Otto and K. Wood. *Product Design*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2001.
- [9] M. H. Salah, T. H. Mitchell, J. R. Wagner, and D. M. Dawson. A smart multiple-loop automotive cooling system—model, control, and experimental study. *IEEE/ASME Transactions on Mechatronics*, 15(1):117–124, 2010.
- [10] C. Zang, M. I. Friswell, and J. E. Mottershead. A review of robust optimal design and its application in dynamics. *Computers & Structures*, 83:315–326, 2005.

# Simulating Rhapsody SysML Blocks in Hybrid Models with FMI

Yishai A. Feldman<sup>1</sup>   Lev Greenberg<sup>1</sup>   Eldad Palachi<sup>2</sup>

<sup>1</sup>IBM Research – Haifa, Israel   <sup>2</sup>Rational, IBM Israel, Rehovot, Israel  
Haifa University Campus, Mount Carmel, 3498825 Haifa, Israel  
{yishai,levg,eldadpal}@il.ibm.com

## Abstract

The Functional Mockup Interface (FMI) standard enables hybrid simulation of models from different tools. Such tools can have different underlying behavioral semantics, creating challenges when models are combined. A case in point is the combination of the Rhapsody tool, widely used to describe and implement discrete control behavior, and Modelica, widely used to describe continuous plant behavior.

This paper describes a plugin we developed for exporting Functional Mockup Units (FMUs) from Rhapsody, and the results of combining generated FMUs with continuous models. When a Rhapsody FMU is used in a different environment, some basic assumptions on its behavior are challenged. We describe the semantic mismatches between the tools, to what extent they can be overcome, and what modelers need to do in order to preserve the intended semantics of an exported FMU.

*Keywords: FMI, Rhapsody, SysML, Hybrid simulation*

## 1 Motivation and Overview

Complex cyber-physical systems are composed out of components and subcomponents, often designed and manufactured by different organizations. Each component can come from a wide range of different engineering domains, including mechanical, electrical, control, and software. Each engineering domain uses its own languages and tools; these are often not integrated with each other. This makes it difficult to perform analysis, verification, and design-space exploration at the model level, resulting in errors that are discovered late in the process (typically during integration), and are expensive to fix.

In some cases there is ad-hoc integration between a pair of tools; for example, organizations commonly create connections using ad-hoc scripts. However, this is expensive, brittle, and sometimes wrong. Func-

tional Mockup Interface (FMI)<sup>1</sup> is an open international standard for the integration of models between different tools that may use different underlying semantics (e.g., discrete state machines and differential algebraic equations). It specifies an interface that encapsulates a model as a *Functional Mockup Unit (FMU)*, for communicating with hosting tools. The standard defines two types of export: *model exchange* and *co-simulation*. The main difference between them is that the former uses a solver of the hosting tool, while the latter includes its own solver and the hosting tool only orchestrates the simulation (transfers variables values, orders FMUs invocation, and selects the next communication step size).

SysML<sup>2</sup> and UML<sup>3</sup> are modeling languages standardized by the OMG to describe structure and behavior of systems at various levels of abstraction. Statecharts [6] are a popular formalism for specifying behavior in SysML and UML; they are an extension of finite-state machines, and enable very concise descriptions of finite-state models. They are used to describe reactive systems at a wide range of abstraction, from the details of embedded controllers to high-level manufacturing processes. Some tools, such as IBM Rational Rhapsody®,<sup>4</sup> can execute statechart models, and, in the case of software applications, to synthesize production code from these models. Model execution (sometimes called “simulation”) can be used to analyze and verify the design.

However, many modern systems are cyber-physical and include physical models whose behavior is modeled in other languages and tools. In the example of Section 2, a statechart is used to model the controller of a heating system; other aspects of the system, such as the thermal plant, sensors, and actua-

<sup>1</sup><https://www.fmi-standard.org>.

<sup>2</sup><http://www.omg-sysml.org>.

<sup>3</sup><http://www.uml.org>.

<sup>4</sup><http://www-03.ibm.com/software/products/en/ratirhapfami>.

tors, are modeled in Modelica®.<sup>5</sup> In order to simulate the whole system, it is therefore necessary to combine the discrete-event simulation of the statechart model in Rhapsody with the continuous dynamics described in Modelica. This paper describes the way in which an FMU that encapsulates the statechart can be exported from Rhapsody and used inside FMI-compliant tools.<sup>6</sup>

In Section 3 we analyze the semantic differences between the behavior of a SysML block in Rhapsody and that of the exported FMU. We highlight subtle, but in some cases significant differences. The naive expectation is for two communicating statecharts from the same Rhapsody model to retain their behavior when each is exported as an FMU and composed in the same way in a hosting tool. Because of the different conceptual semantics of Rhapsody statecharts and the FMI standard, this is very challenging. It follows that designers of SysML models that participate in hybrid simulations as FMUs need to be aware of this and design appropriately. We present a set of guidelines in Section 3.6.

## 2 Example

This section presents an example of a hybrid model, in which a controller specified as a statechart is exported as an FMU and used in a Modelica model. Figure 1 shows a Modelica model of a heater<sup>7</sup> in the SimulationX tool.<sup>8</sup> The model consists of the heater plant (bottom), a temperature sensor (displayed as a thermometer), a controller (top, in black) that receives the sensor input and a parameter specifying the desired temperature, and two actuators that can turn on or off two heating elements (lower left, in blue). A switch (labeled “booleanStep1” in the middle of the diagram) turns the heating system on or off.

The plant is specified in Modelica using a set of differential algebraic equations, and the sensor and actuators are also simple Modelica models. The controller, however, is specified as a SysML statechart in Rhapsody; it is shown in Figure 2. The controller starts in state OFF; it moves to ON when the switch signal changes to true, and moves back when it changes to false.

In state ON, the controller keeps track of the num-

<sup>5</sup><https://www.modelica.org>.

<sup>6</sup>See <https://www.fmi-standard.org/tools> for a list of tools that support FMI.

<sup>7</sup>Based on the ControlledTemperature example from the Modelica Standard Library.

<sup>8</sup><http://www.simulationx.com>.

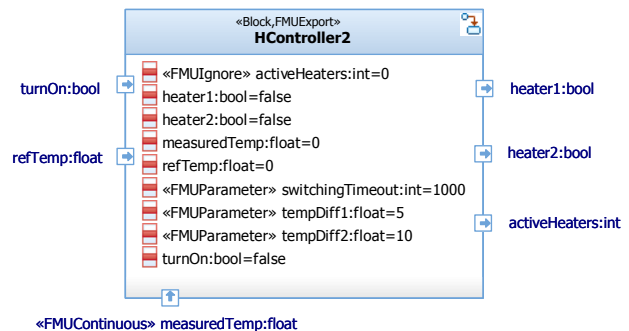


Figure 3: The interface of the controller block.

ber of active heaters (between 0 and 2); in the initial sub-state, NoHeating, this is set to zero, and both heaters are turned off. Each of the three states has a timeout transition (labeled  $tm(\text{switchingTimeout})$ ), it is used to sample the temperature sensor periodically. The operation  $\text{tempDiff}()$  returns the difference between the reference temperature (provided as the top input in Figure 1) and the measured temperature (bottom input). The model has two parameters,  $\text{tempDiff1}$  and  $\text{tempDiff2}$ . Whenever the temperature difference is more than  $\text{tempDiff1}$ , the first heater is turned on; if the difference exceeds  $\text{tempDiff2}$ , both heaters are turned on. This is achieved by the transitions between the states, which check the temperature difference. These transitions have just a condition (specified in brackets) but no trigger. They are checked each time the state is entered; that is, periodically whenever the timeout expires.

The interface of the controller is a SysML *block*, shown in Figure 3. It has three inputs, which are SysML *flow ports*: the boolean-valued  $\text{turnOn}$ , for the on/off switch, and two float-valued input ports, one for the reference temperature, and one for the measured temperature. The last is annotated with the stereotype «FMUContinuous» to indicate that it is a continuous input; by default, inputs are assumed to be discrete (see Section 3.5). Each flow port is associated with a block attribute of the same name (shown inside the block); each of these has an associated initial value. The block has three output flow ports: the first two are boolean-valued signals that turn the heaters on and off, and the third is an integer-valued signal that carries the number of currently active heaters. In addition, the block has three parameters, for the switching timeout and the temperature thresholds.

Figure 4 shows the essential parts of the model description in the FMU generated by the plugin from this block. It starts with a type definition describing the float type associated with the block’s inputs. Since

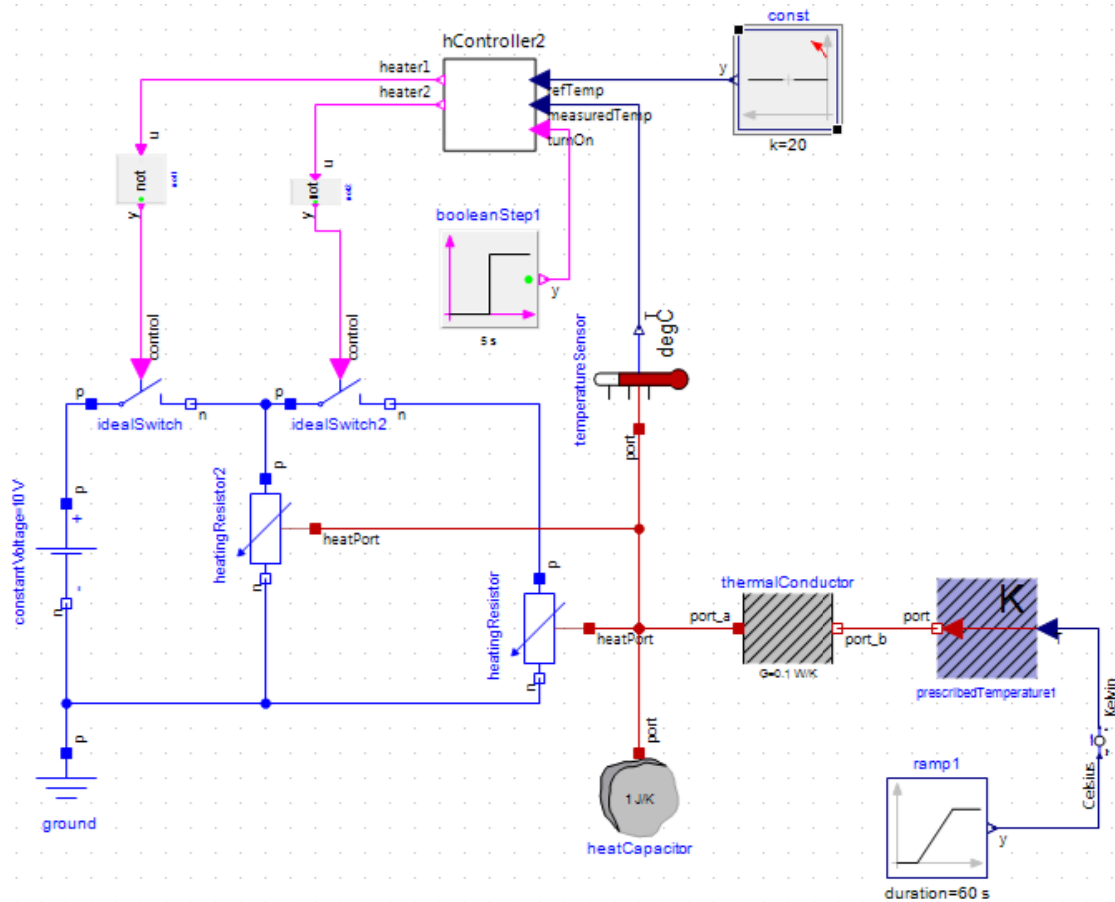


Figure 1: The heater model in SimulationX.

this type defines single-precision floating-point numbers, while the FMI standard defines real numbers using double precision, it is translated into an FMI real type with a constrained range. Following the type definition are the specifications of the model variables. First are the three inputs, the first with a boolean type and the others with the constrained type. All inputs have input causality and discrete variability, except for measuredTemp, which is continuous. Following those are the two output booleans, and the three parameters.

In order to let the plugin know how to treat the various block elements, we defined a UML profile containing a set of stereotypes. For example, the «FMUParameter» stereotype denotes the three block attributes that are to be treated as parameters (Figure 3). The signal activeHeaters is not exported to the FMU, as indicated by the stereotype «FMUIgnore» annotating the block attribute of the same name. The model of Figure 1 defines how these inputs and outputs are connected to the rest of the model when the exported FMU is imported into the SimulationX environment. Inside SimulationX, the imported FMUs are represented as regular Modelica blocks, so they can be naturally in-

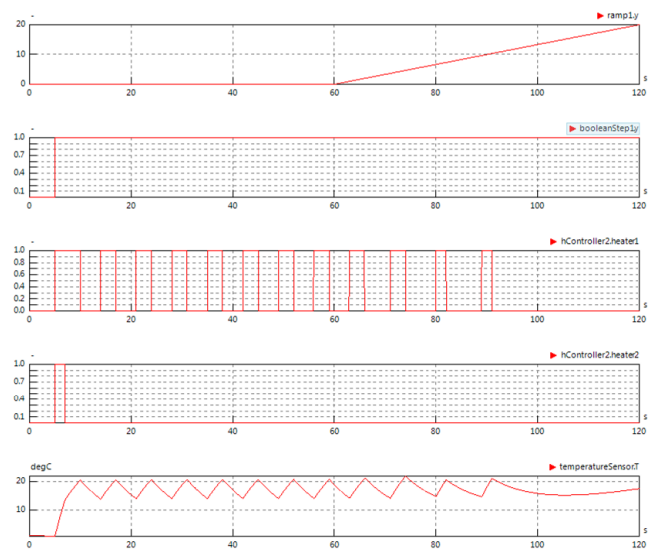


Figure 5: Simulation results with the Rhapsody FMU in SimulationX.

cluded in any Modelica model.

Figure 5 shows the results of the simulation using the exported controller FMU in SimulationX. The top graph shows the ramp1 variable from the block at the

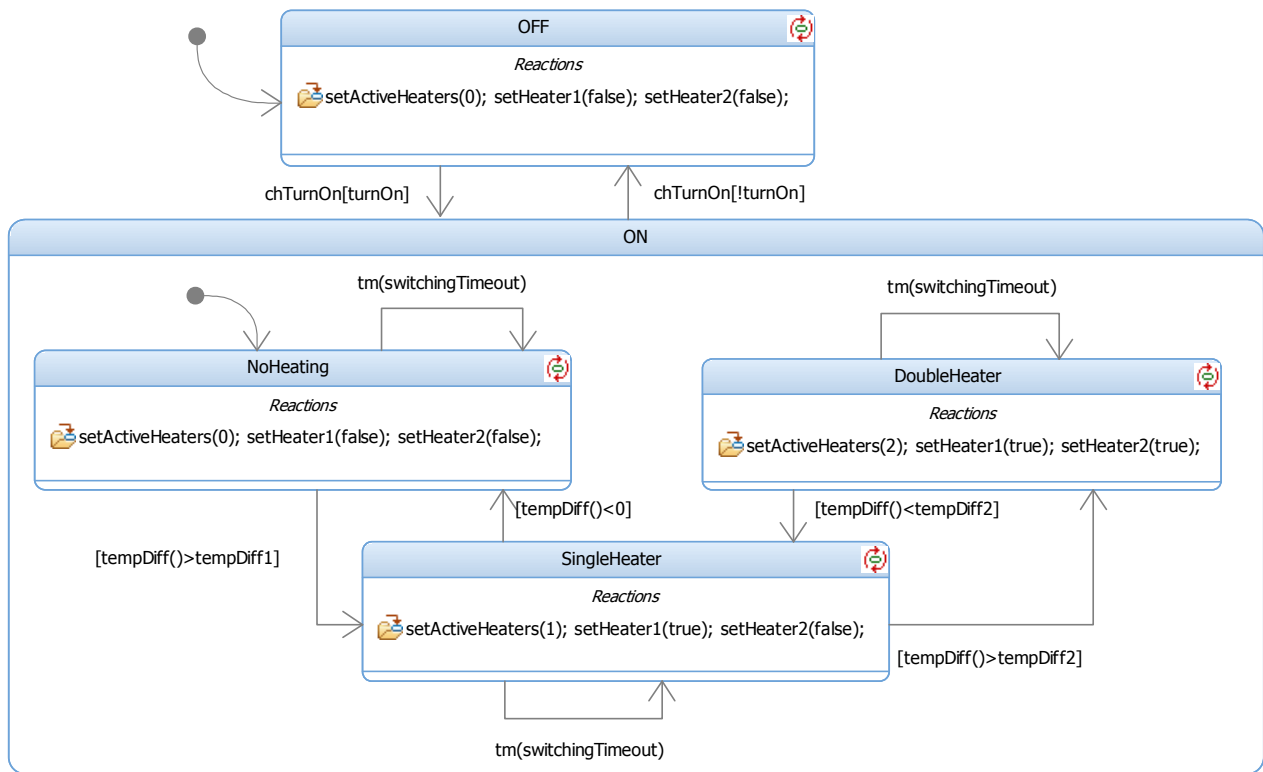


Figure 2: The heater controller in Rhapsody.

top right of Figure 1; this variable represents the ambient temperature, which starts at 0 and climbs to 20 degrees during the second minute. The second graph shows the on/off switch, with is turned on 5 seconds after the simulation starts. The next two graphs show the signals that the controller sends to activate the heaters. When the switch is initially turned on, the measured temperature is zero, and both heaters are activated. The second heater is deactivated after a few seconds, while the first heater is turned on and off according to the measured temperature. The bottom graph shows the values measured by the temperature sensor; the value climbs from 0 to a little over 20 degrees, then seesaws as a result of the activation of the heaters. Toward the end of the second minute, once the ambient temperature is high enough, both heaters are turned off, and the sensed temperature climbs slowly, as expected.

### 3 FMI for SysML

This section describes the FMU export functionality we developed for Rhapsody as a plugin, and discusses the design decisions and their implications. The FMI standard defines two export modes: *model exchange* and *co-simulation*. The major difference be-

tween them is that model exchange requires a hosting tool to provide an ordinary differential equation (ODE) solver to perform the simulation, and therefore the models needs to expose all internal equations. Rhapsody models are discrete and do not need an ODE solver, so co-simulation would be appropriate; this in turn could enable the use of the FMU in environments that do not provide ODE solvers. Unfortunately, FMI for co-simulation cannot handle discrete events efficiently [2]. For example, the co-simulation API, unlike the model exchange API, does not provide a way to report the next event time. We have therefore chosen to use the model-exchange mode for our plugin. In any case, most of the points discussed in this paper apply to co-simulation as well.

The elements of the SysML block that are exported currently limited to atomic flow ports and attributes. As we saw in the example, input flow ports of the SysML block are exposed as inputs of the FMU, and output flow ports become outputs. Values of attributes of the block that have corresponding flow ports are considered by Rhapsody as storage for the values of the ports, and are therefore represented by the same input or output variables. Other attributes will become FMI discrete internal variables; if they are annotated with `«FMUParameter»`, they will become FMI parameters. Initial values of attributes will be translated into

```

<TypeDefinitions>
  <Type name="real32_Type">
    <RealType min="-3.4028234663852886E38" max="3.4028234663852886E38"/>
  </Type>
</TypeDefinitions>
<ModelVariables>
  <ScalarVariable name="turnOn" valueReference="2" variability="discrete" causality="input">
    <Boolean start="false"/>
  </ScalarVariable>
  <ScalarVariable name="refTemp" valueReference="0" variability="discrete" causality="input">
    <Real declaredType="real32_Type" start="0.0"/>
  </ScalarVariable>
  <ScalarVariable name="measuredTemp" valueReference="1" variability="continuous" causality="input">
    <Real declaredType="real32_Type" start="0.0"/>
  </ScalarVariable>
  <ScalarVariable name="heater1" valueReference="0" variability="discrete" causality="output">
    <Boolean start="false"/>
  </ScalarVariable>
  <ScalarVariable name="heater2" valueReference="1" variability="discrete" causality="output">
    <Boolean start="false"/>
  </ScalarVariable>
  <ScalarVariable name="switchingTimeout" valueReference="0" variability="parameter"
    causality="internal">
    <Integer start="1000"/>
  </ScalarVariable>
  <ScalarVariable name="tempDiff1" valueReference="2" variability="parameter" causality="internal">
    <Real declaredType="real32_Type" start="5.0"/>
  </ScalarVariable>
  <ScalarVariable name="tempDiff2" valueReference="3" variability="parameter" causality="internal">
    <Real declaredType="real32_Type" start="10.0"/>
  </ScalarVariable>
</ModelVariables>

```

Figure 4: The `modelDescription.xml` file of the generated FMU (excerpt).

a start value of the corresponding FMU variable. However, the «FMUIgnore» stereotype will prevent any element from being exposed. This gives the user fine control over what parts of the block are to be exposed externally.

Not all SysML elements are currently supported. In particular, the following are challenging, because of mismatches between SysML and the FMI standard:

- Bi-directional flow ports, since the FMI standard does not allow variables that are both inputs and outputs. This restriction may be lifted in future, by artificially creating two variables, one for each direction of a bi-directional port. However, there are various issues such as naming of ports and variables, and synchronizing the values of the input and output variables correctly.
- SysML standard ports, since they carry SysML events (these are not to be confused with FMI

events), which are not supported by the FMI standard.

The FMU export functionality is based on the FMU SDK provided by QTronic,<sup>9</sup> which provides a skeleton implementation of the FMI API. The implementation of the exported FMU consists of the code normally generated by Rhapsody for the block, with an additional wrapper that adapts it to the requirements of the FMI standard. The FMU export process consists of the main steps described in Algorithm 1.

The first step collects the various elements to be exposed according to the rules described above; it also checks for illegal or unsupported combinations and reports them to the user (such reports can be suppressed by using the «FMUIgnore» stereotype). The second step creates the `modelDescription.xml` file, which describes the interface of the generated FMU.

<sup>9</sup><http://www.qtronic.de/en/fmusdk.html>.

---

**Algorithm 1** Generate FMU Wrapper

---

1. Analyze the SysML model to identify input and output ports and internal variables, discover and report errors.
  2. Generate the FMU model description file.
  3. Apply the standard Rhapsody code generation.
  4. Generate the code for the FMU wrapper.
  5. Compile and package the FMU.
- 

---

**Algorithm 2** fmiEventUpdate

---

1. Set Rhapsody time based on the time received in the last fmiSetTime call.
  2. Set the block’s input variables based on the previous set of fmiSetXXX calls.
  3. Invoke the Rhapsody-generated code to execute a behavioral step.
  4. Update the wrapper’s variables corresponding to the FMU outputs.
  5. Set the next event time to the earliest timeout active in the block.
- 

The third step invokes Rhapsody’s code-generation facilities to create the implementation of the behavior of the block to be exported. The fourth step creates the code that adapts Rhapsody’s implementation to the FMI standard. This wrapper code is discussed in most of the rest of this section. Finally, the code is compiled and packaged in the .fmu file.

### 3.1 The FMU Wrapper

The FMU Wrapper generated in step 4 of the FMU export algorithm supports the FMI interface (currently version 1.0 for Model Exchange) and translates it to the Rhapsody interface. The wrapper keeps a set of variables corresponding to the FMU variables, together with a set of other internal variables (for example, the current simulation time). Most of the FMI functions are implemented by reading or setting these variables; the real work is done by the fmiEventUpdate function. The wrapper algorithm implements this function as described by Algorithm 2.

This algorithm seems quite straightforward, but it

hides many subtle semantic issues. When a block is exported as an FMU, its behavior can be changed by the way that the hosting tool delivers variable changes to the wrapper, using the FMI interface calls, and by the way that the wrapper handles these calls. While the former is out of the control of the Rhapsody FMU export plugin, the latter behavior is, and there are a number of different ways to create the wrapper, each of which yields somewhat different semantics. There are two issues that need to be addressed. The first is communication: the way inputs are transferred to the FMU and outputs are received from it. The second is scheduling: when communication takes place, and how much activity the FMU encapsulating the SysML block allows the block to perform before it considers the step to be completed.

Ideally, Rhapsody blocks exported as FMUs would retain their behavioral semantics, and blocks written without consideration for their context could be used as FMUs. At the very least, the naive expectation could be that if two or more Rhapsody blocks are exported as FMUs out of the same model, and are connected in the hosting environment in exactly the same way they were connected in the Rhapsody model, their behavior will not change. However, context does matter, in simulation as well as in physical realizations. Full preservation of semantics between two exported blocks is challenging; see Section 3.2. In the context of non-Rhapsody models, the different semantics of the FMI standard and of Rhapsody create other difficulties, as discussed in Sections 3.3–3.5.

### 3.2 Quoted Out of Context

In this section we consider the case of two blocks from a single Rhapsody model, both of which are exported as FMUs, and connected in the external tool in the same way as they were in the original model. How does the implementation of the wrapper change the semantics of the joint model?

The Rhapsody semantics [7] describes the behavior of a statechart as consisting of a series of steps; each step may consist of several state changes, and may produce several variable-change and other events. There is a strict order between these events.

Changing the value of a SysML flow port in Rhapsody might also create an internal change event (such as chTurnOn in Figure 2); this event can trigger one or more transitions. The value of the attribute corresponding to the port is changed immediately. However, all events in Rhapsody are sent asynchronously; events sent to each port (from whatever source) are



queued, and delivered in order. In particular, updating several variables consecutively creates one event for each, and these are handled one by one. The order is therefore significant; the block may perform arbitrary actions in response to each event; for example, a typical response to an event is for the statechart to move to a new state. Such actions can change how the block responds to the next event.

As mentioned above, the FMI standard allows changing any number of variables in a single event-processing cycle, and the changes are semantically simultaneous. This is a consequence of the synchronous semantics on which the FMI standard is based. A Rhapsody block can perform a sequence of discrete changes in response to a single event. For example, a single transition might include a series of actions that sequentially change a number of outputs. This implies that, in order to ensure that the order of the corresponding Rhapsody events is preserved, these discrete changes must be delivered one by one to the FMU interface. However, this could cause an inconsistency between the values, if several related variables (such as current and resistance) are not changed simultaneously.

The wrapper can implement this strategy by ending step 3 of Algorithm 2 inside each of the setter functions for the output variables in the Rhapsody-generated code, and continuing with steps 4–5. In the next event-processing cycle, the wrapper would allow Rhapsody to continue from the point it was stopped. All of these events, except for the last one, will require another event-iteration cycle by returning `fmiFalse` in the `interactionConverged` field of the return value of the `fmiEventUpdate` function; this does not advance the simulation time.

This strategy allows other FMUs to generate new inputs after each output is reported. These inputs can be delivered by the wrapper to Rhapsody immediately; that would be consistent with the Rhapsody semantics, since it represents an execution of the Rhapsody model in which each block is run in a separate thread and output generation happens to be fully synchronized between threads. However, Rhapsody semantics allows other behaviors; for example, those where one thread is significantly faster than the others.

In this strategy, the exported FMU behaves in accordance with the original semantics of the SysML blocks in Rhapsody. Any strategy that batches consecutive outputs may violate the semantics, since the generated FMU can be used in a context in which the order of these output updates is significant. However,

as mentioned above, the Rhapsody semantics may expose inconsistent value. Furthermore, this strategy could potentially be very inefficient, forcing the whole simulation to receive and react to each discrete change separately. Because of these issues, the current implementation of the FMU wrapper delivers all variable-change events simultaneously; the modeler of the SysML block should be aware of this and introduce small delays between sequential changes when the order is important. This strategy takes the other extreme, in that it allows the block to perform all its possible behavior in a single cycle, and only stops when it reaches a wait for some timeout or event.

A related issue is the treatment of multiple changes to the same variable. In the example, the command to turn the first heater on is given by the signal `heater1`; the command is recognized when the signal changes value from `false` to `true`. It is possible that some path in the statechart contains a series of transitions in the same step, containing the action `setVar(false)` followed by `setVar(true)`. In Rhapsody, both actions will be communicated to the connected element, resulting in two changes in the value of the signal, with two corresponding change events, causing the command to be recognized. However, if this signal is exported as an FMU output, the behavior will depend on how the wrapper treats multiple changes to the same variable. In the implemented version of the wrapper, the second change will override the first, causing the signal to retain its previous value without change. The other block will therefore receive only the last value update, which contains the variable's original value; this might result in different behavior.

In the other strategy, both changes will be delivered separately, and the command will be recognized. A third scheduling strategy, in the middle between these two extremes, is to present changes to different variables together, as in the implemented version, but separate changes to the same variable. This can be done by having the wrapper divide the stream of events received from Rhapsody into segments, each starting with a change to a variable that already has another change in a previous segment. All the events in a single segment will be presented at once, but different segments will be presented separately.

On input (step 2 of Algorithm 2), the wrapper must be ready to accept simultaneous changes of discrete variables, even if it only produces them one at a time, since inputs may be provided by other types of FMUs. There are a number of ways to treat multiple such events. The first is for the wrapper to issue the events

Table 1: Two behaviors of the rounding strategy.

FMI Time ( <i>sec.</i> )	Rhapsody Time ( <i>msec.</i> )	$x_1$	$x_2$	$x_3$	$y$
Scenario 1					
0.0000	0	1	2	3	2
0.0050	5	-1	2	3	2
0.0100	10	1	2	3	2
Scenario 2					
0.0000	0	1	2	3	2
0.0050	5	-1	2	3	2
0.0096	10	-1	2	-3	-2
0.0100	10	1	2	3	-2

to Rhapsody in an arbitrary order, and require modelers of SysML blocks to be exported to ensure that the order does not matter; this is the option currently taken by the plugin. A second option is to impose a specific order on the events; this order can be specified by a SysML stereotype.

### 3.3 A Question of Time

In the current FMI standard, time is measured as a double-precision floating-point value in units of seconds, whereas in Rhapsody time is represented as an integer, in units of milliseconds. This mismatch is another cause for semantic incompatibilities. (The same problem would exist even if the standard used integral units at a different time resolution, such as microseconds.) How should the FMU wrapper handle `fmiSetTime` calls that set the time between two Rhapsody clock ticks?

Of course, a perfect match of the time-lines is impossible. An obvious candidate strategy is to round times to the nearest integral value. However, this strategy causes strange phenomena, where the addition of an unrelated `fmiEventUpdate` call can completely change the behavior. For example, consider a SysML block with three discrete inputs,  $x_1$ ,  $x_2$ , and  $x_3$ , and one discrete output,  $y$ . The behavior of the block is very simple; periodically every 10 milliseconds, it updates  $y$  to be either  $x_2$  or  $-x_2$ , depending on whether  $x_1 > 0$  or not. Input  $x_3$  is not used at all and should not influence this behavior in any way.

Table 1 displays two possible behaviors of this block, under the rounding strategy. In the first scenario,  $y$  is updated at time 0.0, and again at time 0.01. The change of  $x_1$  at time 0.005 does not update the value of  $y$ , since the block is still waiting for its timeout. The second scenario starts in the same way, except that at FMI time 0.0096, variable  $x_3$  gets a new

value. Because this time is rounded to Rhapsody time 10, the block is activated and updates the value of  $y$ . When  $x_1$  is changed back to 1 at FMI time 0.01, the block is already waiting for its next timeout, at (Rhapsody) time 20, and  $y$  is not updated again. The result is a different value for  $y$ , due to a spurious update of  $x_3$ . For continuous systems, due to numerical issues, such differences might be acceptable. For discrete system, working at precise clocks, however, this type of behavior is obviously unacceptable.

Because of this issue, the FMU export plugin uses truncation rather than rounding. With truncation, such undesirable behaviors cannot occur, since intermediate event updates such as the one in the example will not trigger the FMU's timeout and will not advance its internal clock. In the example, FMI time 0.0096 will be translated to Rhapsody time 9, will not trigger the timeout transition, and will not change the value of  $y$ .

It is common in continuous systems to consider a discrete unit of time strictly as a sampling interval, by ignoring all changes that occur between two sampling points except for the last one. However, this approach can result in counterintuitive behavior for discrete models; for example, a slight shift in the timing of a discrete signal change can cause it to be ignored.

### 3.4 Types

A Rhapsody model can employ the full type system of the target language (which, in this paper, we consider to be C). The FMI standard defines a different set of types; these only contain scalar types (real, integer, boolean, string, and enumeration). Each of these can be customized; for example, real and integer types can have an associated range, as shown in the type definition of Figure 4. The FMU export plugin attempts to define the closest possible FMI type for the C type used in the block. Obviously, integral C types are expressed as FMI integers, and floating-point types (`float`, `double`) as FMI reals. Ranges are applied in the FMI types based on the ranges of the C types; however, not all types can be accurately represented in this way. For example, in the 32-bit FMI platform, `fmiInteger` is defined as a C `int`; this means that the C type `unsigned int` has a wider range than that admitted by an FMI integer, which is always signed.

### 3.5 Discrete or Continuous?

Rhapsody is based on a discrete-time model; triggers for transitions between states are all discrete events, and variables are modified in a discrete way (that is,

signals are piecewise constant). Modelers in Rhapsody therefore think of all variables as discrete. However, when a Rhapsody FMU is used in a hybrid model, some of its inputs may be connected to continuous signals. This places restrictions on the way such inputs may be used in the statechart.

Transitions in a statechart are activated by *triggers*, which are discrete events. Triggers include variable-update events (such as `chTurnOn` in Figure 2, which signals a change in the `turnOn` input). The time of a change event for a continuous variable is not well defined, because at a call of `fmiEventUpdate` all FMI variables are updated, whether relevant or not (as in the second scenario of Table 1). This can cause the behavior of the statechart to become unpredictable. Modelers must therefore avoid change events for continuous inputs. The use of continuous inputs in other places is not restricted. For example, checking the value of an input in a transition guard (such as `tempDiff()<0`) does not cause a problem, even if the input is continuous, since the guard is only evaluated at timeout events. The same holds for using a continuous input in an action.

### 3.6 Guidelines for Exportable Blocks

Based on the previous discussion, the following points should be considered when designing a Rhapsody model to be exported as an FMU. First, change events must not be used for continuous signals, although their values can be freely used otherwise. Second, if the order of changes in output variables is important, non-zero delay should be introduced between changes. Third, non-zero delay should be introduced between changes of the same variable if all intermediate values need to be observed. Finally, to preserve communication semantics between Rhapsody blocks or include features that are not yet supported by the FMU export functionality, the composition of the blocks should be exported as a single FMU.

## 4 Related work

Modeling and simulation of hybrid systems is an active research topic [5, 3]. Carloni et al. [4] provide a detailed analysis and comparison of the semantics of commonly-used tools. They conclude that there is a strong need to allow integration of different tools, and suggest leveraging the Hybrid Systems Interchange Format (HSIF) to mediate model semantics between tools. This approach is very different from

the FMI code-generation-based approach used in our work, since FMI is mainly focused on the standardization of a model execution API, and exposes only the model information required for this purpose (for example, whether variables are discrete or continuous, and dependencies between variables).

Other interesting approaches for integrating SysML and Modelica are based on extensions of SysML such as the SysML-Modelica Transformation standard,<sup>10</sup> or of ModelicaML [10] where Modelica could be described using the UML profile extension mechanism. Specifically, Schamai et al. [11] suggest a formal approach to modeling UML statecharts using Modelica, and highlight various semantic differences. For example, in ModelicaML, all available events are processed in parallel in the next evaluation of the state machine, while in Rhapsody statechart events are queued and processed in order according to the UML run-to-completion semantics [7].

In practice, changing languages or tools is a major undertaking, and users are reluctant to do so. Our work therefore concentrates on using the popular Rhapsody tool in new contexts. As discussed in Section 3, this requires some attention to modeling details that might differ in other contexts, but there is no need to change the tool itself. This is similar to the approach taken by Sakairi et al. [9] to integrate Rhapsody and Simulink®, except that they use a proprietary S-function interface.<sup>11</sup> Because we use the FMI standard, our approach is not limited to integration with a single tool or language, but can work with any FMI-complaint simulator.

Pohlmann et al. [8] export FMUs for MechatronicUML [1] instead of Rhapsody SysML. They do not describe any semantic differences between MechatronicUML and the generated FMU. They do say that “a discrete port implements an array of message queues,” since communication in MechatronicUML is asynchronous; it seems, therefore, that the same issues described in Section 3.2 are relevant there as well.

## 5 Conclusions

The FMI standard can be used for hybrid simulation of systems modeled using several tools. We described the Rhapsody plugin that exports FMUs encapsulating SysML blocks whose behavior is defined using statecharts, and demonstrated it on an example that

<sup>10</sup><http://www.omg.org/spec/SyM/1.0>.

<sup>11</sup><http://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html>.

combines the SysML model with a Modelica model. We highlighted subtle but important semantic differences between Rhapsody and FMI, causing the compositional behavior of the FMUs to differ from that of the original blocks, and provided guidelines that will prevent such behavioral differences. Since different tools come with their own semantics, we expect that such mismatches are common, especially when connecting continuous-time with discrete models, and that our guidelines will generalize to many such cases.

## Acknowledgments

We are grateful to Sergey Zolotnitsky for his help with the implementation of the plugin. We thank ITI GmbH for their assistance with SimulationX.

## References

- [1] S. Becker, C. Brenner, C. Brink, S. Dziwok, C. Heinzemann, U. Pohlmann, W. Schäfer, J. Suck, O. Sudmann, and R. Löffler. The MechatronicUML design method – process, syntax, and semantics. Technical Report tr-ri-12-326, Heinz Nixdorf Institute, University of Paderborn, 2012.
- [2] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of FMUs for co-simulation. In *Proc. Int’l Conf. Embedded Software (EMSOFT)*, pages 1–12, 2013.
- [3] D. Broman, E. A. Lee, S. Tripakis, and M. Törn-gren. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *Proc. 6th Int’l Workshop on Multi-Paradigm Modeling*, 2012.
- [4] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1(1–2), 2006.
- [5] P. Derler, E. A. Lee, and A. L. Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.
- [6] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Computer Programming*, pages 231–274, 1987.
- [7] D. Harel and H. Kugler. The Rhapsody semantics of statecharts (or, on the executable core of the UML). In *Integration of Software Specification Techniques for Applications in Engineering*, pages 325–354. Springer, 2004.
- [8] U. Pohlmann, W. Schäfer, H. Reddehase, J. Röckemann, and R. Wagner. Generating functional mockup units from software specifications. In *Proc. 9th Int’l Modelica Conf.*, pages 765–774, 2012.
- [9] T. Sakairi, E. Palachi, C. Cohen, Y. Hatsutori, J. Shimizu, and H. Miyashita. Designing a control system using SysML and Simulink. In *Proc. SICE Annual Conf.*, pages 2011–2017, 2012.
- [10] W. Schamai. Modelica modeling language (ModelicaML): A UML profile for Modelica. Technical report, Linköping University, 2009.
- [11] W. Schamai, U. Pohlmann, P. Fritzson, C. J. J. Paredis, P. Helle, and C. Strobel. Execution of UML state machines using Modelica. In *Third Int’l Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, pages 1–10, 2010.

# Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface

Jonathan Brembeck<sup>1</sup>, Andreas Pfeiffer<sup>1</sup>, Michael Fleps-Dezasse<sup>1</sup>,  
Martin Otter<sup>1</sup>, Karl Wernersson<sup>2</sup>, Hilding Elmqvist<sup>2</sup>

<sup>1</sup>German Aerospace Center (DLR), Institute of System Dynamics and Control,  
82234 Weßling, Germany

<sup>2</sup>Dassault Systèmes AB, Ideon Science Park, 22370 Lund, Sweden

Jonathan.Brembeck@dlr.de, Andreas.Pfeiffer@dlr.de, Michael.Fleps-Dezasse@dlr.de,  
Martin.Otter@dlr.de, Karl.Wernersson@3ds.com, Hilding.Elmqvist@3ds.com

## Abstract

In this paper we propose a method how to automatically utilize continuous-time Modelica models directly in nonlinear state estimators. The approach is based on an extended FMI 2.0 Co-Simulation Interface [1] that interacts with the state estimation algorithms implemented in a Modelica library [2]. Besides a short introduction to Kalman Filter based state estimation, we give details on a generic interface to cooperate with FMUs in Modelica, an implementation of nonlinear state estimation based on this interface, and the Dymola prototype used for the evaluation. Finally we show first results in a tire load estimation application [3] for DLR's robotic electric research platform ROMO [4].

*Keywords: FMI 2.0 Co-Simulation, FMU, Inline Integration, Kalman Filter, State Estimation, Moving Horizon Estimation, Tire Load Estimation*

## 1 Introduction

With the raise of computational power in the last decades the possibilities to implement complex control strategies in real world applications enhanced tremendously. For most of them a good knowledge of the actual states is necessary. Often these are not directly measurable due to cost limitations or missing sensors (for example, it is not practical to measure in-tire forces). In the ITEA2 project MODRIO [5] one aim is to develop state estimation technologies for plants that use the knowledge of complex models of the controlled system itself. These models are often designed, parameterized and optimized as multidomain models in Modelica. To re-use these models for estimation and control purposes the Functional Mockup Interface [1] turns out to be very useful. Three years ago, we presented a concept for state

estimation [2], [6] using FMI 1.0 Model Exchange [7], using Modelica function pointers to separate the prediction model from the observer algorithms and to create an easy reconfigurable framework for state estimation purposes. This approach had several limitations and difficulties that we want to overcome based on a slightly extended FMI 2.0 Co-Simulation Interface [1]. Furthermore we introduced a different way how the user interacts with the prediction model and the desired state estimation method.

One goal of the research performed in MODRIO is to build-up a complete tool chain so that an end-user can utilize a complex model in a state estimation algorithm and download the estimator to an embedded target. To our knowledge such tool chains are not available today. There are toolboxes available for the estimation algorithms, such as [8], [9], but it is non-trivial and time-consuming to utilize them for a concrete application with a nonlinear model and download the result to a real-time target.

The following sections are organized as follows. In Section 2 we briefly recap the well-known Extended Kalman Filter, Unscented Kalman Filter and the Moving Horizon estimator algorithms. From these descriptions we deduce the requirements on a generic interface for nonlinear models. Afterwards, Section 3 gives a closer look into the Modelica implementation, as well as a proposal for a user-friendly configuration interface. Moreover, we show how the FMI 2.0 Co-Simulation needs to be extended to fit the requirements of the prediction steps sketched in Section 2. As a use-case we show an automotive tire load estimation application [3] in Section 4. To validate this approach measurement data acquired with DLR's RoboMObil [4] are used.

## 2 Model Evaluations in State Estimation Algorithms

It is assumed that the plant model to be used in state estimation is naturally described as nonlinear continuous-time state space system:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}), \\ \mathbf{y} &= \mathbf{h}(\mathbf{x}),\end{aligned}\quad (1)$$

$$t \in \mathbb{R}, \mathbf{u}(t) \in \mathbb{R}^{n_u}, \mathbf{x}(t) \in \mathbb{R}^{n_x}, \mathbf{y}(t) \in \mathbb{R}^{n_y}$$

where  $t$  is time,  $\mathbf{u}(t)$  is the vector of inputs,  $\mathbf{x}(t)$  is the vector of states and  $\mathbf{y}(t)$  is the vector of outputs. Such a model shall be provided as a Functional Mockup Unit (FMU) [1]. In this paper plant models are defined in Modelica and exported as FMUs using Dymola. However, all the results are also valid if FMUs are generated by other tools and/or non-Modelica environments, as long as the FMU supports a slightly extended FMI 2.0 Co-Simulation Interface according to our proposal.

Model (1) cannot be utilized directly in a sampled data system. Instead a discrete-time representation is needed for use in a discrete-time state estimator. The following discrete-time version of (1) with additive Gaussian noise is used in the sequel:

$$\begin{aligned}\mathbf{x}_k &= \mathbf{f}_{k|k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1}, \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, \\ \mathbf{w}_k &\sim N(0, \mathbf{Q}_k), \\ \mathbf{v}_k &\sim N(0, \mathbf{R}_k).\end{aligned}\quad (2)$$

Here  $t_k$  is the  $k$ -th sample time instant of a periodically sampled data system,  $\mathbf{u}_k = \mathbf{u}(t_k)$ ,  $\mathbf{x}_k = \mathbf{x}(t_k)$ ,  $\mathbf{y}_k = \mathbf{y}(t_k)$ ,  $\mathbf{w}_k$ ,  $\mathbf{v}_k$  is Gaussian noise, and

$$\mathbf{f}_{k|k-1} = \mathbf{x}_{k-1} + \int_{t_{k-1}}^{t_k} \mathbf{f}(\mathbf{x}, \mathbf{u}_{k-1}) dt. \quad (3)$$

A tool chain has to support (3) because it is non-trivial to transform the natural description (1) in (2).

### 2.1 The Estimation Prediction Step

In this section, we briefly summarize the steps of Kalman Filter based state estimation and will then have a closer look to the prediction step (compare Figure 1) of the Extended Kalman Filter (EKF), the Unscented Kalman Filter (UKF), and a more complex Kalman Filter based algorithm the so-called Moving Horizon Estimation (MHE) [10]. Here the need of an efficient and reliable way for the feed forward model simulation rises tremendously. For further information regarding Kalman Filter techniques, see especially the standard textbook [11]. This section is based on [12], [11] and [2].

In Figure 1 a cycle flow diagram of a recursive Kalman Filter algorithm is depicted. The filter is initial-

ized with the initial state vector guess  $\hat{\mathbf{x}}_0^+$  and the initial guess of the state covariance matrix  $\mathbf{P}_0^+$ . These can be seen as a stochastic expectation for believe in the first guess of the estimation task.

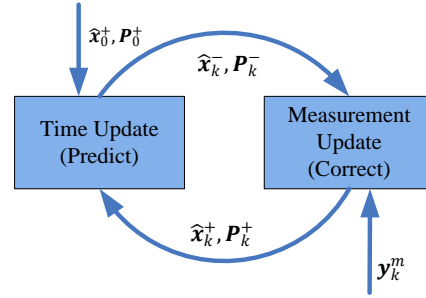


Figure 1: Principle of Kalman Filter based Estimation,  $\mathbf{y}_k^m$  denotes the vector of measured outputs

Afterwards the cycle of the two steps *Predict* and *Correct* begins and is executed with a predetermined static sample time  $T_s$ . The additive Gaussian noise assumption in Eq. (2) is handled by the tuning covariance matrices  $\mathbf{Q}$ ,  $\mathbf{R}$ . These enable the user to tune the filter to the specific task. For nonlinear model state estimation the widely used EKF algorithm is given as pseudo code in Table 1.

Table 1: Extended Kalman Filter Algorithm

<p><i>Initialization:</i></p> $\hat{\mathbf{x}}_0^+ = E(\mathbf{x}_0) \quad // \text{ expectation value}$ $\mathbf{P}_0^+ = E((\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)^T)$ <p>for <math>k = 1, 2, \dots</math>:</p> <p><i>Predict:</i> <math>\hat{\mathbf{x}}_k^- = \mathbf{f}_{k k-1}(\hat{\mathbf{x}}_{k-1}^+, \mathbf{u}_{k-1})</math></p> $\mathbf{P}_k^- = \mathbf{F}_{k-1} \mathbf{P}_{k-1}^+ \mathbf{F}_{k-1}^T + \mathbf{Q}$ <p>where <math>\mathbf{F}_{k-1} = e^{\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{\hat{\mathbf{x}}_{k-1}^+} \cdot T_s}</math></p> <p><i>Correct:</i> <math>\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T \cdot (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R})^{-1}</math></p> <p>where <math>\mathbf{H}_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right _{\hat{\mathbf{x}}_k^-}</math></p> $\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \cdot (\mathbf{y}_k^m - \mathbf{h}(\hat{\mathbf{x}}_k^-))$ $\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \cdot \mathbf{H}_k) \cdot \mathbf{P}_k^-$
--

The red marked sections indicate where the evaluation of the underlying system model equations (2) is necessary. The calculation of  $\hat{\mathbf{x}}_k^-$  is performed by integrating model (1) from  $t_{k-1}$  to  $t_k$ .  $\mathbf{F}_{k-1}$  is the state-transitions matrix of  $\mathbf{f}$  with respect to  $\mathbf{x}$  at  $\hat{\mathbf{x}}_{k-1}^+$  and  $\mathbf{H}_k$  is the partial derivative matrix of  $\mathbf{h}$  with respect to  $\mathbf{x}$  at  $\hat{\mathbf{x}}_k^-$ . The Jacobians  $\mathbf{J}_{k-1}$  and  $\mathbf{H}_k$  must either be provided directly, or they can be determined numerically, for example with a forward difference quotient:

for  $i = 1, 2, \dots, n_x$ ;  $\Delta \cong \sqrt{\epsilon p_s}$

$$(\mathbf{J}_{k-1})_i = \frac{\mathbf{f}(\hat{\mathbf{x}}_{k-1} + \Delta \mathbf{e}_i, \mathbf{u}_{k-1}) - \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1})}{\Delta} \quad (4)$$

## 2.2 UKF Sigma Point Approach

The so called *Sigma Point Transformation* is based on the idea that it is easier to approximate a Gaussian distribution, than it is to approximate an arbitrary nonlinear function or transformation [13], [11]. The parts of the UKF algorithm, where model evaluations are necessary, are given in Table 2. The selection of the *Sigma Points* in matrix  $X$  is performed via a static scaling factor  $\gamma(n, \alpha, \kappa)$  and the matrix square root of the a posteriori covariance matrix. The number of states is denoted by  $n = n_x$ ,  $\alpha$  is the spread around the last state value  $\hat{x}_{k-1}^+$  and  $\kappa$  is a parameter for the stochastic distribution assumption. In total  $2n + 1$  points must be created and then used as initial values for  $2n + 1$  simulations from  $t_{k-1}$  to  $t_k$  to compute  $X_{k|k-1}$ .

Table 2: UKF Prediction Step

$$\begin{aligned}
 X_{k-1} &= \left[ \hat{x}_{k-1}, \hat{X}_{k-1} + \gamma \sqrt{P_{k-1}^+}, \hat{X}_{k-1} - \gamma \sqrt{P_{k-1}^+} \right] \\
 X_{k|k-1} &= f_{k|k-1}(X_{k-1}, u_{k-1}) \\
 \hat{x}_k^- &= \sum_{i=0}^{2n} w_i^m \cdot X_{k|k-1,i} \\
 P_k^- &= \sum_{i=0}^{2n} w_i^c \cdot (X_{k|k-1,i} - \hat{x}_k^-)(X_{k|k-1,i} - \hat{x}_k^-)^T + Q \\
 X_k' &= [\hat{x}_k^-, \hat{X}_k^- + \gamma \sqrt{P_k^-}, \hat{X}_k^- - \gamma \sqrt{P_k^-}] \\
 Y_k &= h(X_k') \\
 \hat{y}_k^- &= \sum_{i=0}^{2n} w_i^m \cdot Y_{k,i}
 \end{aligned}$$

The predicted values  $\hat{x}_k^-$ ,  $\hat{y}_k^-$ ,  $P_k^-$  are calculated via weighted sums with the predetermined weights  $w_i^{c,m}(n, \alpha, \kappa)$ . We define  $\hat{X} := [\hat{x}, \hat{x}, \dots, \hat{x}] \in \mathbb{R}^{n \times n}$  and in our notation a vector depending function (e.g.  $f_{k|k-1}$  or  $h$ ) with a matrix argument returns a matrix with columns that are equal to the evaluated columns of the matrix argument.

It can be shown that the nonlinear approximation accuracy of the UKF is minimum twice higher compared to an EKF. This becomes important in case of strong nonlinearities in the prediction model (for a detailed proof see [14] – Appendix A).

## 2.3 MHE with NLG Method

The *Moving Horizon Estimator* (MHE) is very closely connected to *Model Predictive Control* (MPC). Instead of predicting future control inputs we have a sliding window (with  $M$  steps to the past) that moves every  $T_s$  one step ahead. Therefore, all past  $M$  measurements are taken into account.

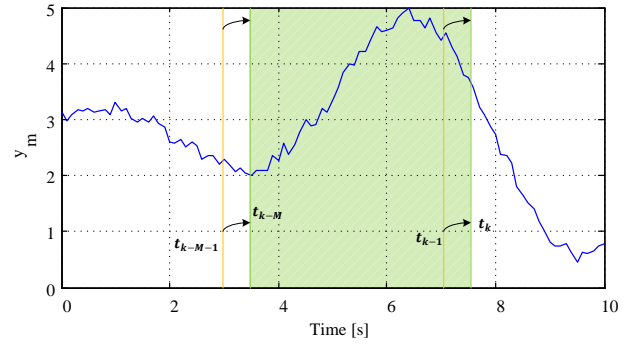


Figure 2: Moving window in MHE application

In this way the estimate gets more robust against external disturbances, delayed measurements can be incorporated and also constraints can be imposed directly [10]. Neglecting the last two points, the optimization objective can be written as follows:

Table 3: MHE Optimization Objective

$$\begin{aligned}
 &\min_{\xi_k} c(\xi_k) \\
 &\xi_k = (x_{k-M}^T, x_{k-M+1}^T, \dots, x_k^T)^T \\
 &x_{int,k-M} = \hat{x}_{k-M}^+ \\
 &x_{int,i} = f_{i|i-1}(x_{int,i-1}, u_{i-1}) \quad (i = k - M + 1, \dots, k) \\
 &c(\xi_k) = \|x_{k-M} - \hat{x}_{k-M}^+\|_{I_{k-M}^+}^2 \\
 &+ \sum_{i=k-M}^k \|y_i^m - h(x_i)\|_{R^{-1}}^2 + \sum_{i=k-M+1}^k \|x_i - x_{int,i}\|_{Q^{-1}}^2
 \end{aligned}$$

The initial constraint (first line in definition of  $c(\xi_k)$ ) is calculated via a Kalman Filter step i.e. an EKF from Table 1, wherein the information matrix  $I_{k-M}^+ = (P_{k-M}^+)^{-1}$ .

For its solution a *Nonlinear Decent Search* (NLG) is useful, because only the first derivatives of the system functions are needed, which is an important constraint for the available interfaces of FMI 2.0 (compare [1]). The algorithm of the unconstrained NLG is given in Table 4, for details please see [15]:

Table 4: MHE Optimization Algorithm

1. Set  $j = 0$  and define  $\xi_k^0 = (x_{int,k-M}^T, \dots, x_{int,k}^T)^T$
2. Decent direction:
 
$$r_j = -\nabla c(\xi_k^j)$$
3. Line search to determine the step size:
 
$$\eta_j = \underset{0 \leq \eta}{\operatorname{argmin}} c(\xi_k^j + \eta \cdot r_j)$$
4. Optimization step:
 
$$\xi_k^{j+1} = \xi_k^j + \eta_j r_j$$
5. If stop criterion not reached:
 
$$j = j + 1 \text{ and go to step 2;}$$

In step 1 an initial solution  $\xi_k^0$  is needed. A good approach for its calculation is the open loop integration of the prediction model from  $x_{k-M}$  to  $x_k$ . The gradi-

ent  $\nabla c(\xi_k^j)$  of the decent direction can be calculated as shown in Table 5 (again, all parts that need a model evaluation are marked in red).

Table 5: MHE Gradient Calculation

$$\mathbf{R}^* = (\mathbf{R} \cdot \mathbf{R})^{-1}; \mathbf{Q}^* = (\mathbf{Q} \cdot \mathbf{Q})^{-1}$$

$$\nabla c_{1:n} = (\mathbf{I}_{k-M}^+ + (\mathbf{I}_{k-M}^+)^T)(\mathbf{x}_{k-M} - \hat{\mathbf{x}}_{k-M}^+) - 2 \frac{\partial \mathbf{h}(\mathbf{x}_{k-M})^T}{\partial \mathbf{x}_{k-M}} \mathbf{R}^* (\mathbf{y}_{k-M}^m - \mathbf{h}(\mathbf{x}_{k-M}))$$

for  $i = k - M + 1, \dots, k$ :

$$\nabla c_{1+(i-k+M):n:(i-k+M+1):n} = \frac{\partial c(\xi_k)}{\partial \mathbf{x}_i}$$

$$= 2\mathbf{Q}^*(\mathbf{x}_i - \mathbf{x}_{int,i}) - 2 \frac{\partial \mathbf{h}(\mathbf{x}_i)^T}{\partial \mathbf{x}_i} \mathbf{R}^* (\mathbf{y}_i^m - \mathbf{h}(\mathbf{x}_i))$$

## 2.4 Summary of Needed Model Evaluations

In Table 6 all needed evaluations of the prediction model (compare Eq. (1), (2)) for state estimation applications are summarized. A tool chain has to provide these model evaluations. In the right column the name of the Modelica function is listed to trigger the corresponding evaluation in the tool chain proposed by this article, for details see section 3.3.

Table 6: Model evaluations for nonlinear state estimation (in the right column the name of the Modelica functions are defined to trigger the evaluations in the tool chain proposed in this article, see section 3.3).

Required model evaluations		Modelica
Integration between two sample points:	$\mathbf{f}_{k k-1} = \mathbf{x}_{k-1} + \int_{t_{k-1}}^{t_k} \mathbf{f}(\mathbf{x}, \mathbf{u}_{k-1}) dt$	integrator
Derivative evaluation:	$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$	f
Output evaluation:	$\mathbf{y} = \mathbf{h}(\mathbf{x})$	h
<b>Optional model evaluations</b> (if not provided, computed numerically by difference quotients)		
State Jacobian matrix:	$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{u})$	fx
Output Jacobian matrix:	$\frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x})$	hx

## 3 Nonlinear Kalman Filters and FMI

In Section 2 different state estimation algorithms are summarized. The goal of this paper is to provide a tool chain for nonlinear state estimation based on the

model equations of a Modelica model. In [2] it is explained why a pure Modelica solution to reach this goal is currently not possible. Using FMI helps to overcome this situation. In [2] we concentrated on FMI 1.0 for Model Exchange with the drawback that the integration algorithm for performing prediction steps of a Kalman Filter has to be implemented in Modelica, a non-trivial task. FMI 2.0 for Co-Simulation [1] simplifies the implementation significantly because the integration algorithm, including event handling, is embedded inside the FMU. Still some features are missing. In a Dymola prototype these have been added in order that the “required” functions from Table 6 are supported.

The overall process of using a state estimator in Modelica is illustrated in Figure 3:

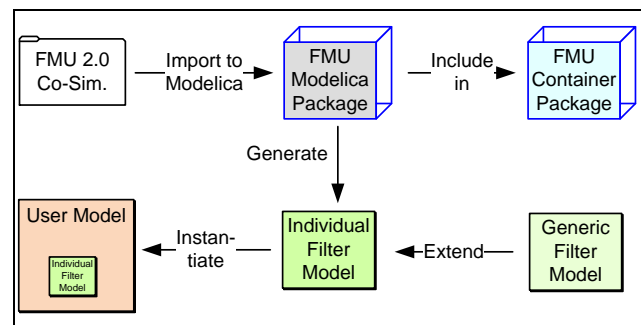


Figure 3: Process flow to generate a state estimator based on an FMU

An FMU (usually exported from a Modelica model) is imported into the Modelica environment by extending the package `FMUImportTemplate`. The imported package can be included in an FMU container package to collect several FMUs for easy access. For such an FMU package an *Individual Kalman Filter* model is generated that provides variable names on buses and user convenient parameter menus. The algorithmic part of the state estimation is provided in a *Generic Filter Model*. Finally, the individual filter model can be instantiated in the user’s application model. An example is shown in the next figure:

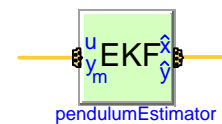


Figure 4: Instance of an individual EKF Kalman filter model generated by the process from Figure 3. The bus on the left side contains the individual input and measurement variables of the FMU and the bus on the right side contains the individual estimated state and output variables of the FMU.

In the following sub-sections, the details of the process from Figure 3 are described.



### 3.1 FMI for State Estimation with Dymola

The standard FMI Co-Simulation Interface allows integrating (1) from sample instant  $t_{k-1}$  to  $t_k$  with function `fmiDoStep(..)` and therefore computing (2). In standard co-simulation the continuous-time states of a model are hidden in the co-simulation slave. However, for state estimation the states need to be explicit and it must be possible to reset the states at sample instants, see Section 2. In order to achieve this, Dymola 2014 FD01, that has already support for FMI 2.0 Co-Simulation according to [1], has been extended in a prototype with the needed features. Especially,

- the continuous-time states are reported in the `modelDescription.xml` file under element `ModelStructure`,
- it is possible to explicitly set the continuous-time states with `fmiSetReal(..)` before `fmiDoStep(..)` is called,
- it is possible to inquire the actual values of all variables with `fmiGetReal(..)` after `fmiSetReal(..)` was called, without an `fmiDoStep(..)` in between,
- when importing an FMU for Co-Simulation in to Modelica, Dymola generates the Modelica code optionally according to the `FMUImportTemplate` package shown in the next section. This package serves as interface to access the needed FMI functionality from a Modelica model or function.

### 3.2 FMUImportTemplate package

Importing an FMU means to generate a package that contains all the functionality needed to simulate the FMU or use it in a state estimator. For this the template package `FMUImportTemplate` is provided, see code and figure below. The imported FMU extends from the `FMUImportTemplate` and redeclares all elements.

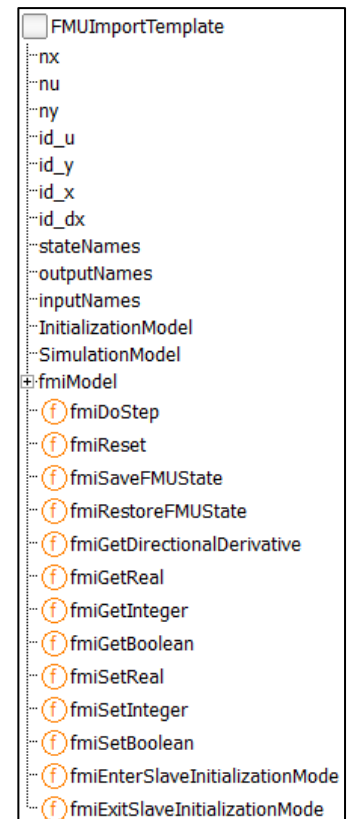
```
partial package FMUImportTemplate
  constant Integer nx=1;
  ...
  constant Integer id_x[nx];
  ...
  constant String stateNames[nx];
  ...
  replaceable model SimulationModel
  end SimulationModel;

  replaceable model InitializationModel
    fmiModel fmi;
    parameter Real fmiInitOk(fixed=false);
  end InitializationModel;
```

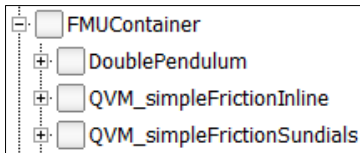
```
replaceable partial class fmiModel
  extends ExternalObject;
  function constructor
  ...
  end constructor;
...
end fmiModel;

replaceable function fmiDoStep
  input fmiModel fmi;
  ...
  end fmiDoStep;
...
end FMUImportTemplate;
```

Important dimensions of the FMU such as the number of continuous states `nx`, inputs `nu` and outputs `ny` are set in the imported FMU package. Furthermore, the FMI references are available by the vectors `id_x`, `id_dx`, `id_u` and `id_y` for state, state derivative, input and output variables. It is also important to get variable names for states, inputs and outputs. Otherwise the order of the components in the vectors `x`, `u`, `y` would be only visible by user-unfriendly reference values instead of variable names. The names are used in the parameter GUIs of the filter model in the next subsection and in the input and output bus of a filter model.



The imported FMU package contains two models: `SimulationModel` and `InitializationModel`. The model `SimulationModel` is a fully operating Modelica model (with inputs and outputs) that wraps the FMU for Co-Simulation whereas in `InitializationModel` only the FMU is instantiated by the external object `fmiModel` and the FMI initialization phase is executed. The `InitializationModel` is used in a Kalman Filter model; the `SimulationModel` is contained for completeness to use the imported FMU package also for other applications like a “real” FMU for Co-Simulation in the Modelica simulation environment.



The FMU package provides interface functions to all (or at least most) of the functions defined in

the FMI Co-Simulation standard 2.0. For a user-convenient handling of the FMU import process, it is desirable to import an FMU as a sub-package into an existing Modelica package. The default package is the package `FMUContainer` that hosts several imported FMUs, see figure above.

### 3.3 Model Functions for State Estimators

The state estimator algorithms are implemented with Modelica functions that provide the needed model evaluations. In partial package `BaseFunctions` the interfaces of these functions are defined and in package `SystemFunctions` the function prototypes are collected. The latter are replaceable functions that provide the needed functionality of Table 6 (the right column of this table lists the name of the function).

For example, partial function `fBase` is defined as:

```

partial function fBase "Base class of the
state equation dx/dt = f(x,u,t)"
  input Integer nx "Number of states";
  input Integer nu "Number of inputs";
  input Real x[nx] "States";
  input Real u[nu] "Inputs";
  input Modelica.SIunits.Time t "Time";
  output Real dxdt[nx] "Derivatives";
end fBase;
  
```

The dimensions `nx`, `nu` are conceptually not necessary, because the dimensions could be determined by the size of the vectors `x` and `u`. Currently, Dymola does not support arrays with non-fixed sizes in function calls of translated Modelica models. The function prototypes are collected in package `SystemFunctions`:

```

partial package SystemFunctions
  replaceable function f
    extends fBase;
  end f;
  ...
  replaceable function integrator
    extends integratorBase;
  end integrator;
end SystemFunctions;
  
```

For a particular model, an implementation of the `SystemFunctions` functions has to be provided. For FMUs, this is performed with the generic package `FMISystemFunctions`. The implementation is based on the `FMUImportTemplate` package and holds therefore for every FMU that extends from this template package.

```

package FMISystemFunctions
  extends SystemFunctions;
  replaceable package FMU
    constrainedby FMUImportTemplate;
  redeclare function extends f
    input FMU.fmiModel fmi;
  algorithm
    FMU.fmiSetReal(fmi, FMU.id_u, u);
    FMU.fmiSetReal(fmi, FMU.id_x, x);
    dxdt := FMU.fmiGetReal(fmi, FMU.id_dx);
  end f;
  ...
  redeclare function extends integrator
    input FMU.fmiModel fmi;
  algorithm
    FMU.fmiSaveFMUState(fmi);
    FMU.fmiSetReal(fmi, FMU.id_u, u);
    FMU.fmiSetReal(fmi, FMU.id_x, x);
    FMU.fmiDoStep(fmi, t, dt, 0);
    xNew := FMU.fmiGetReal(fmi, FMU.id_x);
    FMU.fmiRestoreFMUState(fmi);
  end integrator;
end FMISystemFunctions;
  
```

The system functions `f`, `h`, `integrator` can be directly implemented with functions provided in `FMUImportTemplate`. The Jacobians `fx` and `hx` are implemented by computing them numerically with finite difference quotients. Once Dymola supports directional derivatives for imported FMUs for the extended Co-Simulation case, that is function `fmiGetDirectionalDerivatives`, then this function can be directly called and will provide a more efficient and reliable evaluation of the Jacobians.

The Dymola prototype supports two techniques for the FMI function `fmiDoStep`. Either the *Sundials* solvers [16] are used (that are integrators with variable step size and error control) to numerically integrate the model equations, or *Inline integration* [17] is applied, that means fixed step solvers are embedded in the model equations. The Kalman Filter library works with both techniques. For real-time applications, fixed-step methods have to be used and therefore a Kalman filter will usually utilize *Inline integration*.

The functions `fmiSave/RestoreFMUState` in the above code fragments are auxiliary functions that call the FMI functions `fmiGet/Set/FreeFMUState` to enable several calls of `fmiDoStep` starting at the same time instant, as needed, for example, for the UKF.

### 3.4 Tailored Kalman Filter Models in Modelica

Based on the imported FMU package an individual Kalman Filter model has to be generated. In the current version of the Kalman Filter Library this can be performed automatically by use of a Modelica/Dymola scripting function. The idea is to define an input bus `InBus` and an output bus `OutBus` for

exchanging variables between the filter model and higher level models. The names of the bus variables correspond to the variable names of the imported FMU – only “:”, “;”, “[”, “]” and “ ” are replaced by “\_” due to Modelica syntax. The bus definitions for the use-case example in Section 4 are listed below:

```
encapsulated expandable connector InBus
import Modelica;
extends Modelica.Icons.SignalBus;
// Model Inputs
Real u;
// Measured Model Outputs
Real accBody;
Real sRel;
Real accArmUp;
end InBus;

encapsulated expandable connector OutBus
import Modelica;
extends Modelica.Icons.SignalBus;
// Estimated Model States
Real mass_wheel_s;
Real mass_wheel_v;
Real mass_body_s;
Real mass_body_v;
Real ...FirstOrderShapingFilter_s;
// Estimated Model Outputs
Real accBody;
Real sRel;
Real accArmUp;
end OutBus;
```

The advantage of this approach is that not vectors of anonymous variables are defined, but bus variables with meaningful names tailored to each individual FMU. The main state estimation algorithms are implemented in sub-functions and in a partial filter model, e.g. for an UKF (see Section 2.2). This model defines several variables and parameters for the filter algorithm that is called at each sample point of a sampled integration time interval. In the filter model also an instance of `InitializationModel` of the imported FMU package is included. Together with the package `FMISystemFunction` all necessary parts are put together to run FMI based Kalman Filter algorithms within a Modelica model.

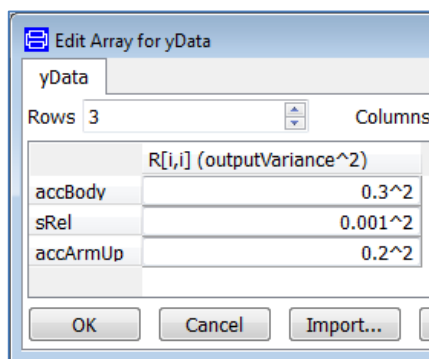


Figure 5: Parameter menu for output variances with names of output variables

A further improvement of the user interface compared to [2] are the filter parameters like state and

output variances that are shown in lists with names of the respective variables – instead of indices of vectors, see Figure 5. Basically, a matrix is defined and via Dymola specific annotations row and column headings can be added to the parameter menu. For example the menu in Figure 5 is defined in the following way:

```
parameter Real yData[FMUPackage.ny,1]
annotation(Dialog(
  __Dymola_columnHeadings =
    {"R[i,i] (outputVariance^2)"},
  __Dymola_rowHeadings =
    {"accBody", "sRel", "accArmUp"}));
```

In the parameter menu of the filter in Figure 6 the user can press the button on the right side of `yData` to get the menu of Figure 5. Also the model parameters of the FMU may be modified by clicking on the button on the right side of `ModelParameters`.

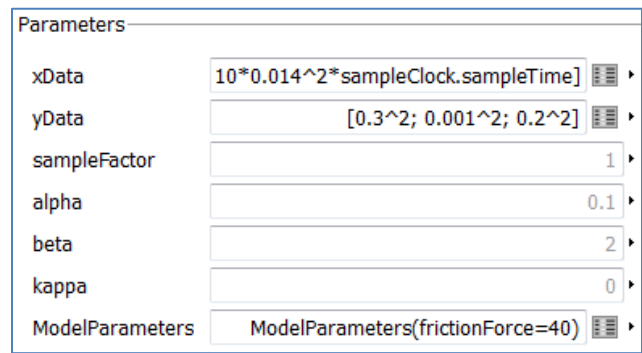


Figure 6: Menu of a UKF SR Kalman Filter model

## 4 Example: Vehicle Vertical Dynamics States Estimation

As an example application, the above described Kalman Filter Library is used to develop an advanced state estimator for the vertical dynamics of the ROboMObil. A more detailed version of this application case is available in [3]. This section is based on [3], but now the method and software from Section 3 are used. The ROboMObil is a robotic electric vehicle concept (see Figure 7) developed at the Robotics and Mechatronics Center of the German Aerospace Center DLR. It is comprised of four Wheel Robots (Figure 8 – left), which integrate traction motor, steering, brake system, spring and semi-active damper. Further details on the ROboMObil can be found in [4].



Figure 7: ROMO on the four-post test rig

In contrast to fully active suspension systems they need far less energy [19]. Therefore a diversity of control strategies for semi-active dampers is presented in literature. An overview about these control strategies can be found in [20] and [21]. As most of these strategies need state feedback, but not all states can be measured, state estimation becomes an important topic during the design of semi-active suspension systems.

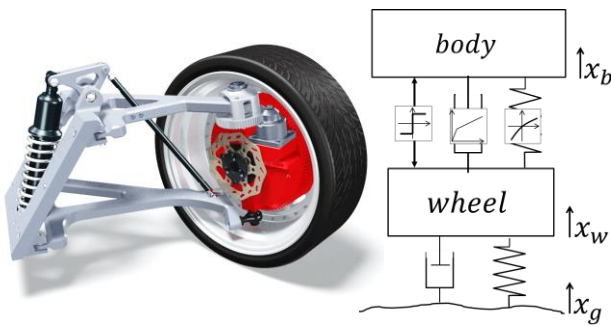


Figure 8: Left: the “Wheel Robot” concept, right: nonlinear two mass system [3]

The DLR *Kalman Filter Library* as presented in Section 3 offers an easy to use framework for the development of state estimators for nonlinear systems like this semi-active suspension system. Especially a square root utilizing implementation of the UKF (Subsection 2.2) algorithm SR-UKF is well suited for highly nonlinear systems, because of its higher order linearization accuracy of mean and covariance. Furthermore the nonlinear parts can be taken into account more easily than in an EKF algorithm (Subsection. 2.1) since no derivatives and Jacobians are needed.

#### 4.1 The Nonlinear Quarter Vehicle Model

The suspension system of the Wheel Robot is modeled as a nonlinear two mass system (see Figure 8 – right) as described in [3]. The corresponding implementation in Modelica is shown in Figure 9. The model consists of the two masses *mass\_body* and *mass\_wheel*, a linear spring damper component, which approximates the wheel behavior, a road mod-

el as explained in [18] or [22] and the *body\_spring* and *body\_damper*.

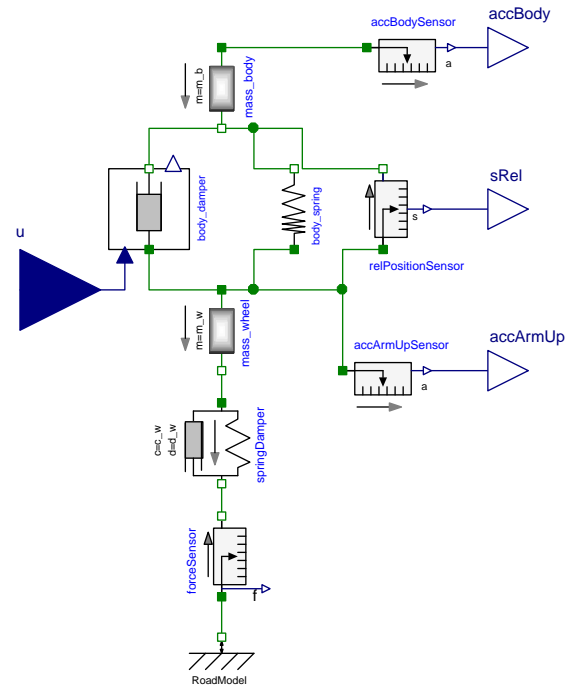


Figure 9: Nonlinear two mass system in Modelica

As the motion of these two components are connected to the wheel and body motion by a push rod-rocker kinematic (compare Figure 8 – left), the motion and the force of these components is scaled by a transmission ratio. Details on the nonlinear characteristic of the *body\_damper* are shown in [3].

The third main nonlinearity of the two mass system besides the transmission ratio and the damper characteristic is the friction of the suspension system. It covers the friction of the damper and of all joints of the suspension system. For the state estimation the friction force  $F_f$  is modeled without stiction by a smooth tanh-switching function:

$$F_f = F_{f,const} * \tanh(v_d / \epsilon_f). \quad (5)$$

Here  $F_{f,const}$  represents a constant sliding friction. The direction of the friction force is determined according to the current velocity difference  $v_d$  between body and wheel. The parameter  $\epsilon_f$  is used to define the transitional behavior of the tanh-function.

#### 4.2 Experimental Setup and Results

The nonlinear two mass system described in Section 4.1 is integrated in an SR-UKF state estimator using the DLR Kalman Library including the FMI 2.0 for Co-Simulation interface and Inline-integration as described in Section 3. Subsequently the resulting estimators are applied to the measurement data recorded with the ROboMObil on a four-post test rig. Figure 10 shows the Modelica model of the SR-UKF

estimator. On the left-hand side the measurement data is read by a *CombiTimeTable* and on the right-hand side the estimator, called *Filter*, and its corresponding settings block *observerControl* can be found. The estimator uses three measurement inputs: the acceleration of the body above the wheel, the wheel acceleration and the damper deflection.

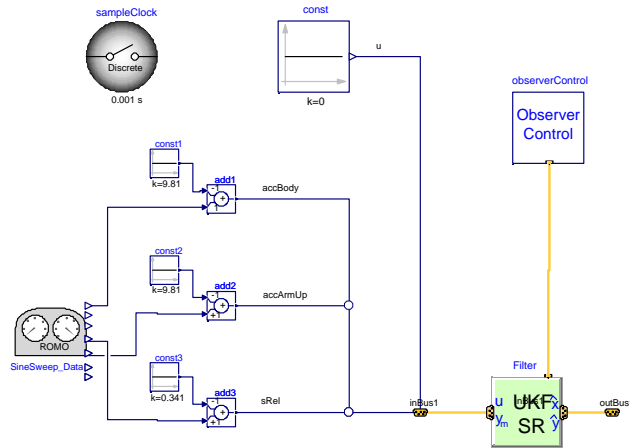


Figure 10: SR-UKF estimator in Modelica

The parameters of the estimators, as well as the system covariances, are tuned according to the optimization procedure presented in [3]. The measurement covariances are set according to the sensor noise. The experimental setup is shown in Figure 11.

The performance of the estimator, subject to a sine sweep excitation, is shown in Figure 11 by comparing the measured and the estimated tire contact forces in the last plot. Please notice that the tire force  $F_{z_{measure}}$  is only available on the four-post test rig (Figure 7). It is used for validating the estimator performance and not as a measurement output  $y^m$  to it (compare experimental setup in Figure 10). Additionally the measurements are compared to the estimated measurements. It can be seen that the estimator reproduces the measurements and the tire contact force with a good accuracy.

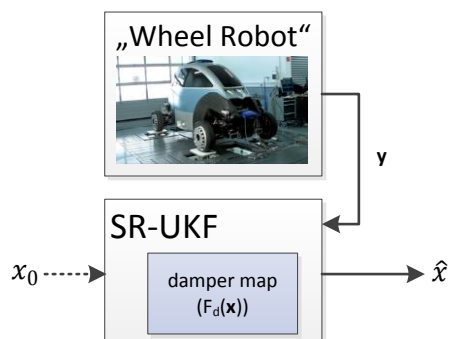


Figure 11: State estimator setup with measurement data from the four-post test rig

As the body accelerometer has the largest noise level, the weighting of its measurements *accBody* was chosen in such a way that the estimator relies more

on the damper deflection *sRel* and the wheel acceleration *accArmUp*.

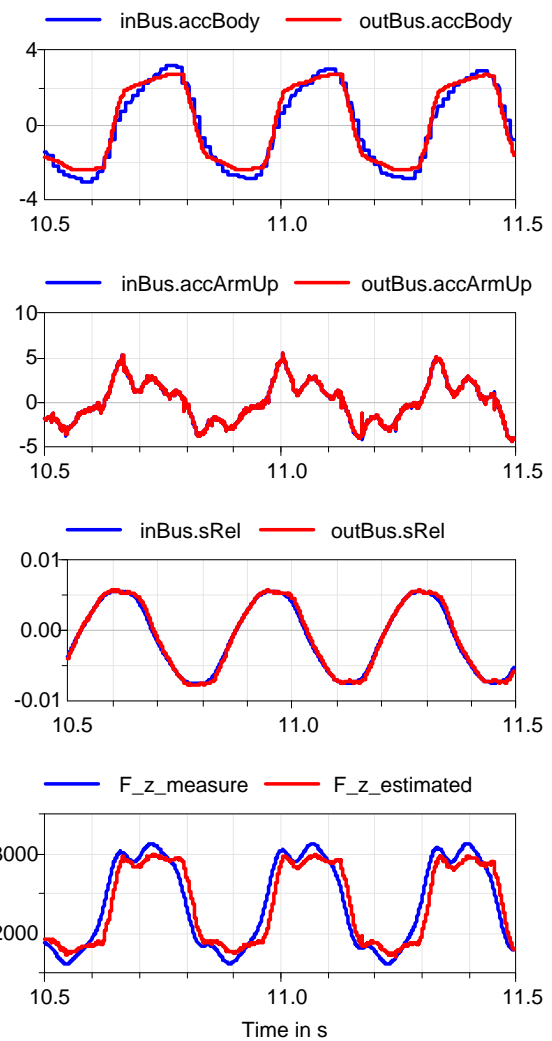


Figure 12: Comparison of measurements to estimated measurement (plot 1-3) and contact force estimation (plot 4) on the four-post test rig – “sine sweep” excitation

## 5 Conclusions and Outlook

In this paper we have shown how the FMI 2.0 standard for Co-Simulation [1] can be extended to use its capabilities for modern state estimation problems. We discussed the parts of the different estimation algorithm in detail, the needed evaluation of the system functions and integration between two sample instants via the FMI interface. Furthermore, we gave implementation details on a user friendly workflow as well as a set of necessary Modelica models and functions. The use case example of our vertical dynamics state estimation [3] application for our RoboMobil [4] showed good results in accuracy and computational efficiency. As a next step we plan to test them on a commercial real-time platform for stability and deterministic execution.

## 6 Acknowledgement

This paper is based on research performed within the ITEA2 project MODRIO. Partial financial support of the German BMBF and the Swedish VINNOVA for this development are highly appreciated.

Furthermore, the authors would like to thank Bernhard Thiele from DLR for his support to implement and cross-compile FMUs on real-time hardware and Marcus Baur for his support in implementing Kalman Filter algorithms in Modelica.

## References

- [1] **Functional Mockup Interface for Model Exchange and Co-Simulation 2.0 RC1**, Modelica Association, 18.10.2013. [Online]. Available [Accessed 25.1.2014]: <https://www.fmi-standard.org/downloads#version2>.
- [2] J. Brembeck, M. Otter and D. Zimmer, **Nonlinear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles**, in *Proceedings of 8th International Modelica Conference*, Dresden, 2011. Available [Accessed 25.1.2014]: <http://www.ep.liu.se/ecp/063/053/ecp11063053.pdf>.
- [3] M. Fleps-Dezasse and J. Brembeck, **Model based vertical dynamics estimation with Modelica and FMI**, in *IFAC ACC*, National Olympics Memorial Youth Center, Tokyo, Japan, 2013.
- [4] J. Brembeck, L. M. Ho, A. Schaub, C. Satzger, J. Tobolar, J. Bals and G. Hirzinger, **ROMO - the robotic electric vehicle**, in *22nd IAVSD International Symposium on Dynamics of Vehicle on Roads and Tracks*, Manchester, 11.-14. Aug. 2011.
- [5] **Modrio ITEA2**, [Online]. Available [Accessed 25.1.2014]: <https://modelica.org/external-projects/modrio> <http://www.itea2.org/project/index/view/?project=10114>.
- [6] C. Engst, **Object-Oriented Modelling and Real-Time Simulation of an Electric Vehicle in Modelica**, Masters Thesis, TUM EAL Munich, 2010.
- [7] **Functional Mock-up Interface for Model Exchange, Version 1.0**, 25 January 2010. [Online]. Available [Accessed 25.1.2014]: <https://www.fmi-standard.org/downloads#version1>.
- [8] J. Hartikainen and S. Särkkä, **Optimal filtering with Kalman filters and smoothers - A Manual for Matlab toolbox EKF/UKF**, February 2008. [Online]. Available [Accessed 25.1.2014]: <http://www.lce.hut.fi/research/mm/ekfukf/>.
- [9] B. Houska, H. J. Ferreau and M. Diehl, **ACADO toolkit – An open-source framework for automatic control and dynamic optimization**, *Optimal Control Applications & Methods*, vol. 32, no. 3, pp. 298-312, 2010.
- [10] D. Simon, **Kalman filtering with state constraints: a survey of linear and nonlinear algorithms**, *IET Control Theory & Applications*, vol. 4, no. 8, p. 1303+, 2010.
- [11] D. Simon, **Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches**, 1. Edition ed., Wiley & Sons, 2006.
- [12] J. Brembeck, **Method to extend models for system design to models for system operation**, MODRIO First Project Report, Munich, 2013.
- [13] S. J. Julier and J. K. Uhlmann, **Unscented filtering and nonlinear estimation**, *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401-422, March 2004.
- [14] S. Haykin, **Kalman Filtering and Neural Networks**, Wiley-Interscience, 2001.
- [15] J. Rosen, **The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints**, *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 1, pp. 181-217, 1960.
- [16] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban and D. E. Shumaker, **SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers**, *ACM Transactions on Mathematical Software*, vol. 31(3), pp. 363-369, 2005. Software: Available [Accessed 25.1.2014] <http://computation.llnl.gov/casc/sundials/main.html>.
- [17] H. Elmqvist, F. Cellier and M. Otter, **Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems**, in *European Simulation Multiconference*, Prague, 1995. Available [Accessed 25.1.2014] [http://www.inf.ethz.ch/personal/fcellier/Pubs/OO/esm\\_95.pdf](http://www.inf.ethz.ch/personal/fcellier/Pubs/OO/esm_95.pdf).
- [18] M. Mitschke and H. Wallentowitz, **Dynamik der Kraftfahrzeuge**, Berlin: Springer, 2004.
- [19] R. Williams, **Electronically controlled automotive suspensions**, *Computing Control Engineering Journal*, vol. 5, no. 3, pp. 143-148, jun 1994.
- [20] S. Savaresi, C. Poussot-Vassal, C. Spelta, O. Sename and L. Dugard, **Semi-Active Suspension Control Design for Vehicles**, Elsevier Science, 2010.
- [21] E. Guglielmino, **Semi-active suspension control - Improved vehicle ride and road friendliness**, London: Springer, 2008.
- [22] G. Koch, T. Kloiber, E. Pellegrini and B. Lohmann, **A nonlinear estimator concept for active vehicle suspension control**, in *Proc. American Control Conf. (ACC)*, Baltimore, 2010.

# Model-based Development of Future Small EVs using Modelica

Yutaka Hirano    Shintaro Inoue    Junya Ota  
 Toyota Motor Corporation, Future Project Division  
 1200 Mishuku, Susono, Shizuoka, 410-1193 JAPAN  
 {yutaka\_hirano, shintaro\_inoue\_aa, junya\_ota}@mail.toyota.co.jp

## Abstract

To cope with demands for future low carbon society, development of new-type small electric vehicles (EVs) becomes very active. To reduce the energy consumption in various actual driving conditions, considering overall running resistance such as aerodynamic resistance, tire rolling resistance including cornering drag, mechanical and electrical losses, etc. will be necessary. On the other hand, to cope with reduced stability against external disturbances such as side wind because of the light weight, it was clarified that additional control of direct yaw moment is effective. In this paper, model-based development of a new electric vehicle using Modelica is described. Full vehicle model considering both vehicle dynamics and energy consumption was developed and utilized to investigate the best possible solutions for both basic design of the vehicle and design of the control system.

*Keywords: Future electric vehicles; Stability and Handling Performance; Energy Consumption*

## 1 Introduction

To cope with future mobility society, development of many new concept vehicles is increasingly active in recent years. Figure 1 shows a new EU regulation about light weight vehicles [1]. Those vehicles have characteristics of smaller size, lighter weight, less number of passengers than conventional vehicles. Also those vehicles tend to be equipped with lower RRC (Rolling Resistance Coefficients) tires and new driving systems mainly using electric motors to achieve less emission and less energy consumption. On the other hand, Toyota has a vision about future eco-cars as shown in Figure 2. Toyota thinks EVs are

suitable as short-distance mobility though there is a possibility of extending the driving range using range extender devices such as small combustion engine, additional battery and so on. In this paper, model-based-development of a new vehicle using Modelica is described. The models were developed based on Vehicle Dynamics Library (VDL) of Dymola.





Category & Category Name	Sub category & Sub category name	Example
L6e, Light quadricycle	L6Ae Light on-road quad	
	L6Be Light mini-car	
L7e, Heavy quadricycle	L7Ae Heavy on-road quad	
	L7Be Heavy mini-car	

Figure 1: New EU Regulation "Light-category vehicles" [1]

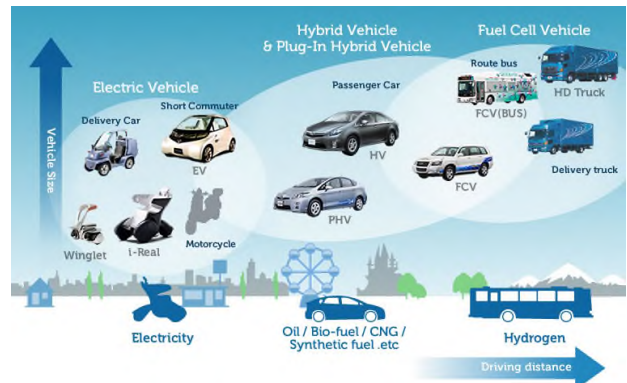


Figure 2: Toyota's scenario about future eco-cars

## 2 Modeling and simulation studies of the future vehicle

### 2.1 Target vehicle

Table 1: Specifications of the target vehicle

	Target plan	L7Be
Vehicle weight	< 600 kg	< 400* kg
Passengers	4	2
Max. Payload incl. passengers	300 kg	200 kg
Rated power	25 kW	< 15 kW
Max. speed	120 km/h	> 45 km/h
Driving range	> 100 km	-

(\* weight without batteries)

Table 1 shows comparison of specifications between our plan and EU regulation L7Be. Our aim is to develop a heavier and more powerful vehicle with more passengers than L7Be considering actual usefulness.

### 2.2 Simulation studies about basic specifications

To consider energy consumption, handling, stability, ride comfort and NVH (noise, vibration, harshness) performances of holistic vehicle, a full-vehicle model including mechanics, electronics, vehicle dynamics and control was made using Dymola. Moreover, a model of a new drive train system such as torque vectoring differential gear was developed and connected into the full-vehicle model. Figure 3 shows an example of the full-vehicle model. Details of the model will be explained later.

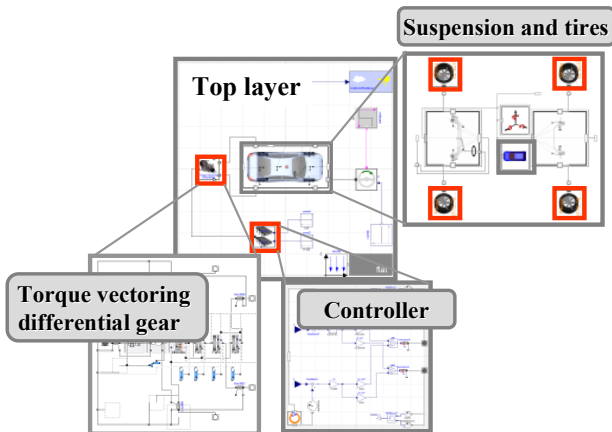


Figure 3: An example of full-vehicle model

Power consumption of each system was calculated simultaneously and was used for the investigation of good balance of energy consumption and vehicle performances. At first, total power of resistances acting on the vehicle was calculated by following equations [2].

Total resistance power:

$$P_v = P_{rr} + P_{ar} + P_{sy} + P_{sx} \quad (1)$$

Rolling resistance power:

$$P_{rr} = \mu_r mg \times V \quad (2)$$

Aerodynamic resistance power:

$$P_{ar} = \rho AC_D V^2 / 2 \times V \quad (3)$$

Cornering resistance power:

$$P_{sy} = \left[ \left( \frac{d_f}{C_{pf}} + \frac{d_r}{C_{pr}} \right) \times mA_y^2 / g \right] \times V$$

$$\approx \left[ \left( \frac{1}{C_p} \right) \times mA_y^2 / g \right] \times V \quad (4)$$

Longitudinal resistance power:

$$P_{sx} = (mA_x + mg \sin \theta) \times V \quad (5)$$

Here

$\mu_r$ : rolling resistance coefficient (RRC),

$g$ : acceleration of gravity [ $m/s^2$ ],

$m$ : vehicle mass [kg],

$V$ : vehicle speed [m/s],

$\rho$ : air density [ $kg/m^3$ ],

$A$ : vehicle frontal area [ $m^2$ ],

$C_D$ : aerodynamic resistance coefficient,

$d_f$ : front weight distribution ratio,

$d_r$ : rear weight distribution ratio,

$C_{pf}$ : front normalized cornering power [1/rad],

$C_{pr}$ : rear normalized cornering power [1/rad],

$C_p$ : average normalized cornering power [1/rad],

$A_y$ : lateral acceleration [ $m/s^2$ ],

$A_x$ : longitudinal acceleration [ $m/s^2$ ],

$\theta$ : road inclination [rad].



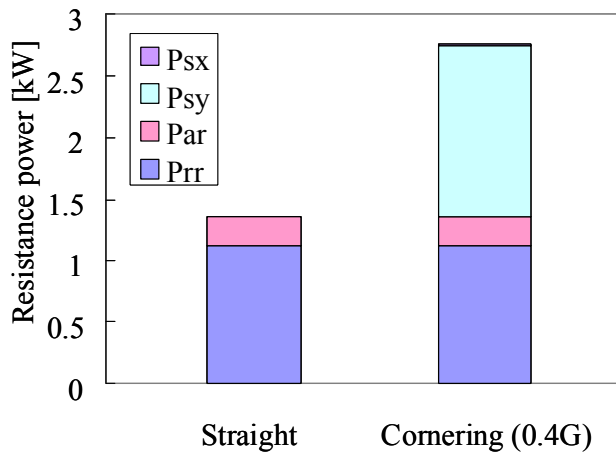


Figure 4: Comparison of resistance powers while driving straight and cornering ( $V = 30[\text{km/h}]$ ) by simulation

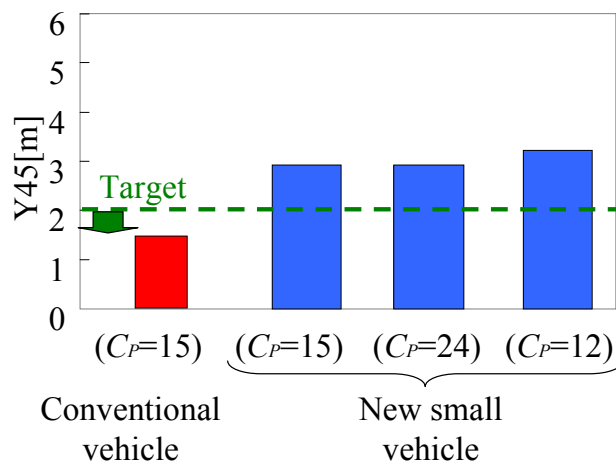


Figure 5: Simulation results of parameter study for stability against side wind between conventional vehicle and the new small vehicle

Figure 4 shows an example of a comparison of resistance powers between straight driving and cornering. It became clear that rolling resistance and cornering resistance were rather big and reducing those resistances was essential to reduce the power consumption. From equations (2), (4) and (5), it is understood that decreasing vehicle mass and tire RRC and also increasing tire cornering power (CP, normalized by tire contact load) are effective to reduce the total resistance power and improve the energy consumption of the vehicle. However, in general, decreasing RRC tends to result in decrease of CP for ordinary tires. Moreover, it is expected that decreasing vehicle mass

will result in reduced vehicle stability against external disturbances such as side wind. Figure 5 shows a result of parameter study for evaluating stability against side wind by Dymola. Evaluation criteria ( $Y_{45}[\text{m}]$ ) in Figure 5 is the lateral deviation while driving at 120 km/h and pass a zone of 45m length with the side wind of 20m/s. In Figure 5, comparison between conventional vehicle ( $m = 1050[\text{kg}]$ ) and the new small vehicle ( $m = 600[\text{kg}]$ ) are shown. Also for the new small vehicle, some levels of normalized CP were researched. It became clear that the light-weight small vehicle is affected more than the conventional vehicle by side wind, and sensitivity of normalized CP value against the disturbance is very small. From above investigations it is indicated that developing new tire which can realize both low RRC and high CP value is necessary for reducing energy consumption. Also for coping with improving vehicle stability against external disturbances for such small vehicles, additional control of vehicle dynamics such as direct yaw moment control is considered to be necessary.

### 3 Development of necessary items to improve holistic performance of the new small vehicle

#### 3.1 New suspension system using tires with low RRC and high CP value

As mentioned in the previous section, the development of new tires for which both low RRC and high CP value can be realized will be necessary for reducing overall energy consumption. For this purpose, a new concept of tires called Large and Narrow (L&N) Concept was developed by Bridgestone [3]. It has characteristics of larger overall diameter, narrower section width and higher inflation air pressure than conventional tires. Figure 6 (cited from [3]) shows comparison of RRC and CP measurement data between ordinary reference tire, L&N tire and L&W (Large and Wide) tire representing current high performance tire. It can be seen that L&N tire has good balance of lower RRC and higher CP at high inflation pressure of 320 kPa. Thus, it was decided to adopt L&N tires for our new vehicle. Figure 7 shows a comparison of overall cornering resistance force ( $Cr$ ) calculated by the equation (6) when running on a constant radius corner at lateral acceleration of 0.4G.

Tire size	LI	OD [mm]	Category
175/65R15	84	608	Reference
205/50R18	89	660	Large & Wide
155/55R19	76	656	Large & Narrow

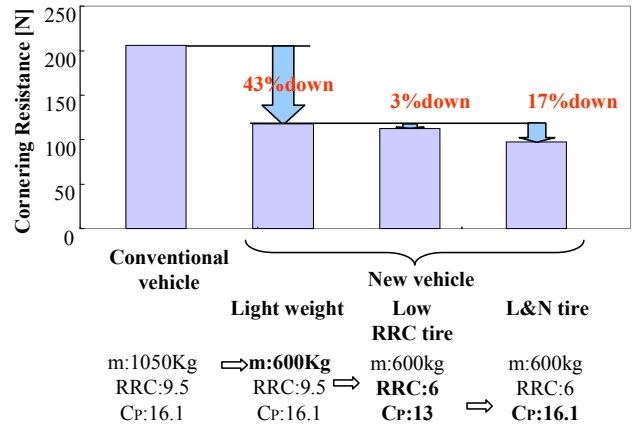
  


Figure 7: Comparison of cornering resistance force by simulation

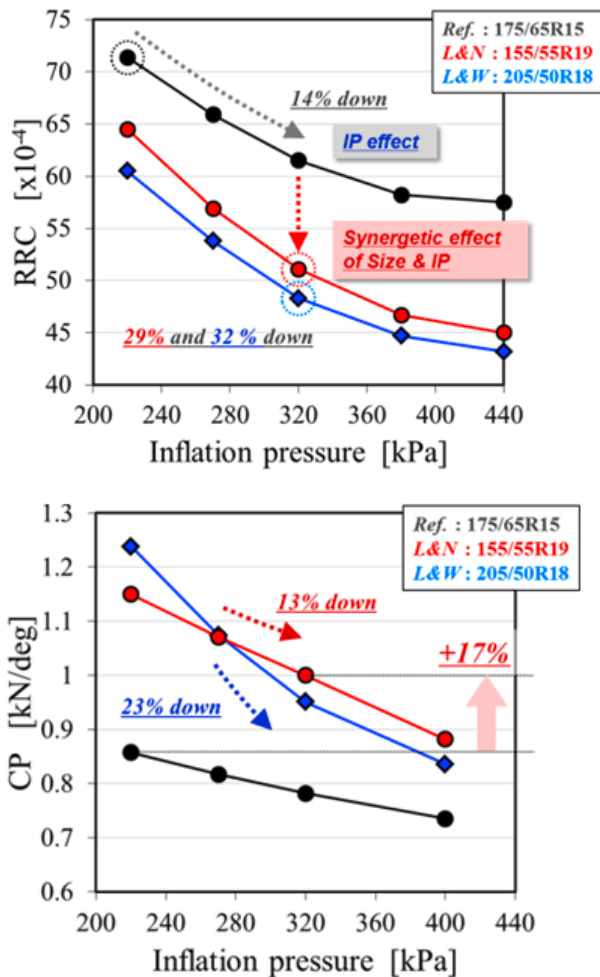


Figure 6: Measurement data of RRC and CP at Load=3.5kN [3]

$$Cr = mA_y^2 g / C_p \text{ [N]} \quad (6)$$

Comparison is done between conventional vehicle and new small vehicle with ordinary tire, low RRC (but low CP) tire and L&N tire. The effect of low weight and L&N tire to reduce the cornering resistance and thus energy consumption was proved by the simulation.

However, there still are remaining problems for applying L&N tires for the new vehicle. Because of larger overall size, it has larger rotating inertia resulting in larger drive-train vibration and less controllability of driving torque than conventional vehicle. Also higher inflation pressure results in higher vertical stiffness and it is suggested to affect ride comfort and NVH (Noise, Vibration and Harshness). Thus, new suspension design is supposed to be necessary. To cope with those problems are future works.

### 3.2 Active yaw moment control by torque vectoring system

As mentioned above, it became clear that active yaw moment control to cope with external disturbances was indispensable for small light-weight vehicles. To research the best solution of this function, benchmarking of existing torque vectoring systems were performed using simulation by Dymola at first. Considering the space for mounting and also controllability, TUM:MUTE type system[4] was investigated further. In this system, a main motor connected to outer ring gear of the differential planetary gear set controls total driving torque. On the other hand, a control motor connected with an input shaft of control gear sets controls torque distribution of left and right

wheel. The function of torque distribution was confirmed by Dymola simulation as shown in Figure 8. It became clear that torque distribution ratio can be changed from 50:50 to both of 0:100 and 100:0 and more by increasing the input torque of the control motor. By this simulation, also energy consumption of both main motor and control motor was able to calculate as well as mechanical transient motion. Finally an example of desired yaw rate feedback control for the torque distribution ratio was tested. Main motor torque ( $T_m$ ) was decided by PI feedback control of difference between desired value and actual value of the vehicle speed by following equation.

$$T_m = K_{pv} \cdot (V_{ref} - V) + K_{iv} \cdot \int (V_{ref} - V) dt \quad (7)$$

where

- $K_{pv}$ : Proportional feedback gain
- $K_{iv}$ : Integral feedback gain
- $V_{ref}$ : Desired vehicle speed
- $V$ : Actual vehicle speed

On the other, control motor torque ( $T_c$ ) was calculated by PI feedback control of difference between desired yaw rate and actual yaw rate as following equation.

$$T_c = K_{pr} \cdot (r_{ref} - r) + K_{ir} \cdot \int (r_{ref} - r) dt \quad (8)$$

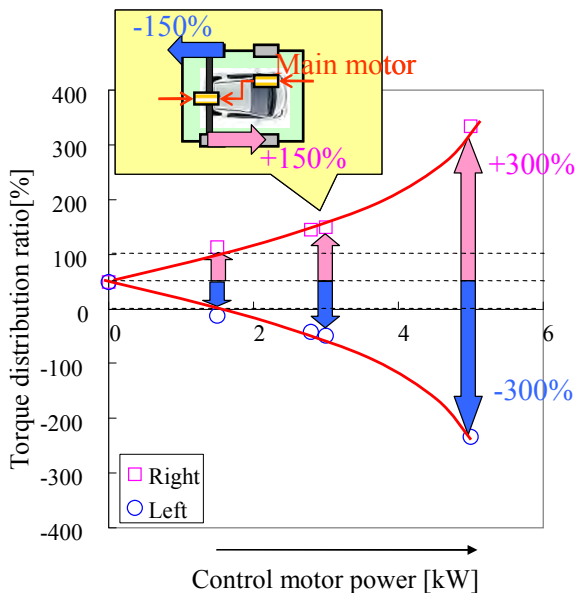


Figure 8: Simulation result of torque distribution ratio VS control motor power for torque vectoring differential gear system of MUTE type

where

- $K_{pr}$ : Proportional feedback gain
- $K_{ir}$ : Integral feedback gain
- $r_{ref}$ : Desired yaw rate
- $r$ : Actual yaw rate

and the desired yaw rate was calculated as below.

$$r_{ref} = \frac{K_s}{1 + sT_s} \delta_s \quad (9)$$

$$K_s = \frac{(a_f + a_r)c_f c_r V}{a_r c_r m V^2 + a_f (a_f + a_r)c_f c_r} \quad (10)$$

$$T_s = \frac{m a_f V}{(a_f + a_r)c_f} \quad (11)$$

Here,

- $\delta_s$ : Steering input angle at front tire
- $a_f$ : Longitudinal distance between front wheel and CG (Centre of gravity)
- $a_r$ : Longitudinal distance between rear wheel and CG
- $c_f$ : Cornering stiffness of front two tyres
- $c_r$ : Cornering stiffness of rear two tyres.

Figure 9 shows a simulation result of the side wind test. Lateral deviation against the side wind (Y45) and energy consumption of main motor and control motor were calculated for the cases of no control, only P feedback control and PI feedback control. The contradiction between vehicle stability and energy consumption of the control motor was confirmed. Finally, development of a new torque vectoring differential gear based on parallel planetary gear sets was decided. Numerical consideration as above will enable us to design the best solution for the practical design of the systems both in mechanical aspect and electrical aspect.

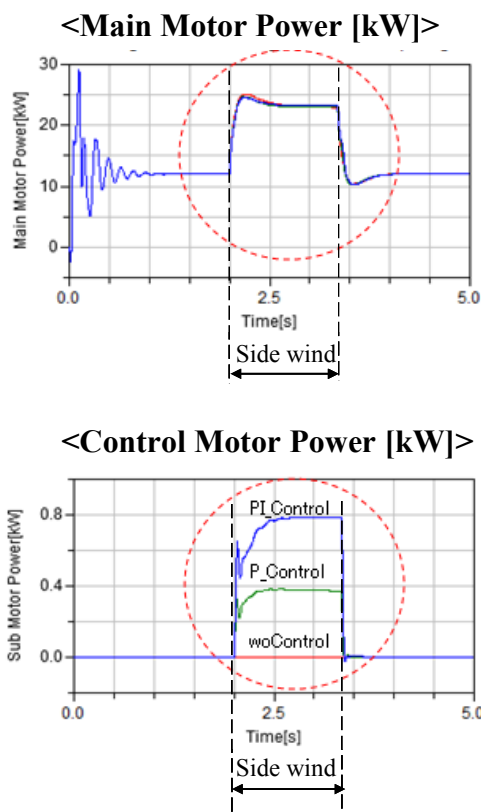
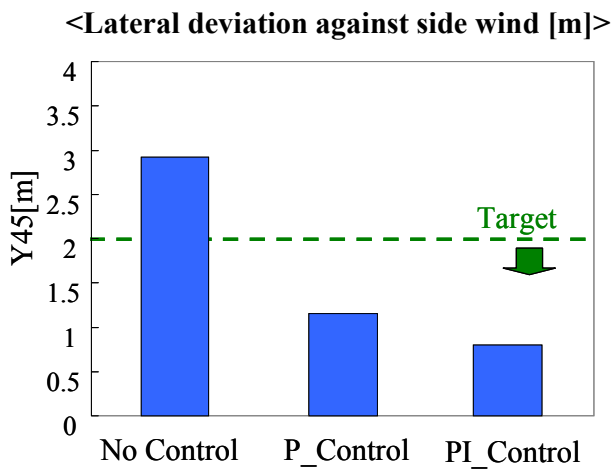


Figure 9: Simulation results of side wind test between No control, P feedback control and PI feedback control of torque vectoring differential gear

### 3.3 Electric regeneration system of braking force

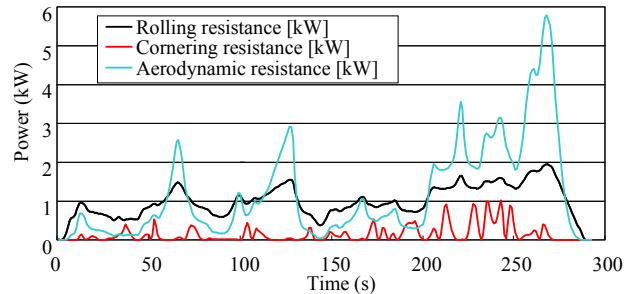


Figure 10: An example of calculation of each resistance power when driving on a winding circuit road

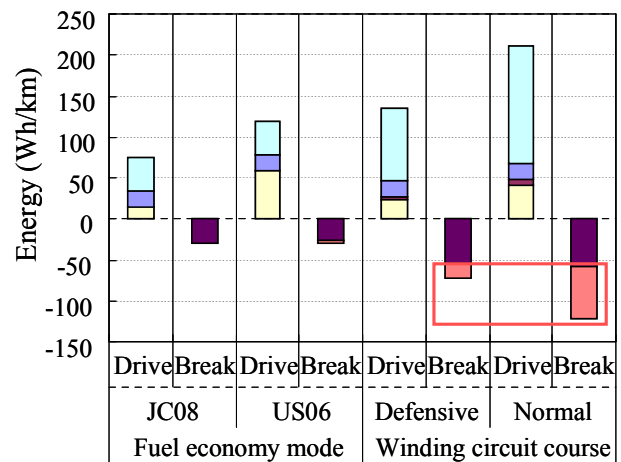
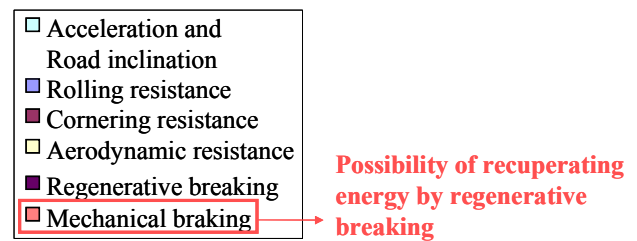


Figure 11: Comparison of driving and braking energy for some examples of driving modes

Utilizing electric regeneration of braking force by motor is effective to improve the energy consumption. In the planning phase of a new vehicle, it is necessary to decide proper size of battery capacity for the regeneration. For this purpose, realistic simulation of actual driving scenes is necessary. IPG CarMaker was used to calculate the vehicle speed, longitudinal and lateral acceleration, road decline and corner radius for actual roads. Using above data, necessary driving power and overall resistance power by the equation

(1) were calculated. Figure 10 shows an example of resistance power of each resistance force while driving on a winding circuit road. Using these results, overall driving energy and braking energy for some examples of driving mode were calculated as shown in Figure 11. Here, JC08 and US06 are regulation of driving modes for measuring fuel consumption in Japan and USA respectively. Figure 12 shows distribution of vehicle speed and longitudinal acceleration for both modes. As seen in Figure 12, US06 mode uses higher vehicle speed and acceleration than JC08 mode and results in more driving power and braking power appearing in Figure 11. Also in Figure 11, comparison of two driving styles for a winding circuit road is shown. It is observed that the defensive driving style needs less power than the normal driving style. It is assumed that braking power less than 15kW can be recuperated by the regeneration of motor in this example. It was understood that there is remained braking power which can be recuperated if the regeneration ability of battery system is large enough in the case of circuit road driving. In this example of Figure 11, these braking powers are consumed by the mechanical breaks and wasted. Figure 13 shows a result of simulation to calculate possible electricity consumption value in the cases of using battery systems whose regeneration ability are 15kW, 40kW and 70kW respectively. It became possible to estimate how large battery capacity was necessary to improve the energy consumption in each driving cases.

Consequently it was proved that these simulations were very useful to decide the proper specifications of a new vehicle in the planning phase.

## 4 Conclusions

For the investigation of overall vehicle specifications and system structures, a holistic vehicle model including mechanics, electrics, electronics, vehicle dynamics and control was made and utilized. It was proved that such a holistic model was very useful to investigate the proper specifications and system constructions in the phase of early stage of vehicle development. Modelica was very suited to make such a multi-discipline and multi-domain investigation by model-based development.

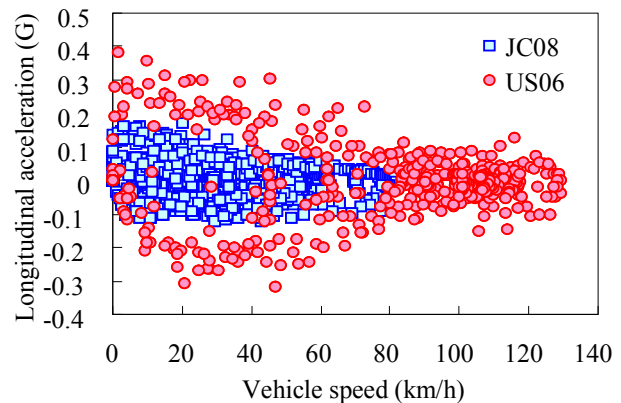


Figure 12: Vehicle speed VS longitudinal acceleration for JC08 and US06 modes

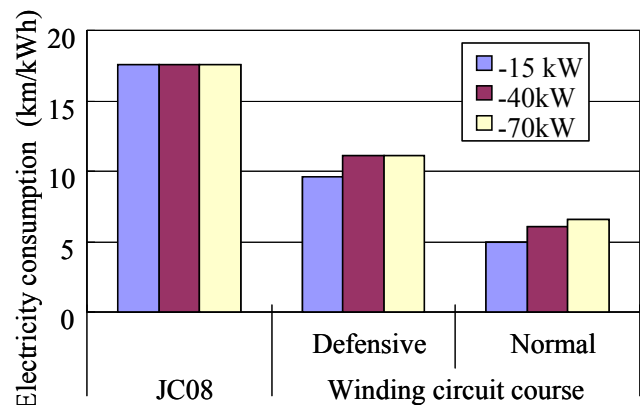


Figure 13: Comparison of estimated electricity consumption when changing the battery's ability of regeneration

## References

- [1] Informal document GRB-55-15 (55th GRB, 7-9 February 2012, agenda item 7(b)), European Commission, Enterprise and Industry
- [2] Kobayashi T., Katsuyama E., Sugiura G., Ono E., Yamamoto M.: "A research about driving force distribution control and energy consumption while cornering", Proceeding of 2013 JSAE Annual Congress (Spring), 352-20135393, 2013 (in Japanese)
- [3] Kuwayama I., Matsumoto H., Heguri H.: "Development of a next-generation-size tire for eco-friendly vehicles", Proceeding of

- Chassis.tech plus | 4th International Munich Chassis Symposium 2013, pp.623-644, 2013
- [4] Höhn B., Stahl K., Wirth C., Kurth F., Lienkamp M., Wiesbeck F.: “Electromechanical Power Train with Torque Vectoring for the Electric Vehicle MUTE of the TU München”, *Getriebe in Fahrzeugen 2011 Effizienzsteigerung im Antrieb*, 7./8./ Juni (2011), Friedrichshafen. VDI-Berichte Nr. 2130, 2011

# Modeling of an Electric Axle Drive with Modelica: A Study of Electric Active Dynamics

Hasan Flaih Awad                      Frank Rettig                      Tomas Smetana  
Schaeffler Technology AG & CO. KG  
Industriestrasse 1-3, 91074 Herzogenaurach, Germany  
{awadhsa, rettifan, smetatma}@schaeffler.com

## Abstract

This paper focuses on modeling of the electric axle drive system (eAxle) used for improving vehicle stability and handling performance by means of a torque vectoring (TV) feature as well as improving vehicle traction and reducing CO<sub>2</sub> emissions by means of an electric traction feature. The function, construction and benefits of the eAxle will be explained within these contexts. An overview of the modeling of the eAxle in Dymola® will be shown. Several simulation cases are conducted to verify the effectiveness of the system for reducing fuel consumption and improving longitudinal and lateral dynamics. A co-simulation was developed between Dymola® and Abaqus® to simulate the power loss and ascertain the temperature behavior on the housing of the eAxle. Finally, the eAxle with a vehicle model was driven over a special realistic handling course using an open source software called Blender®. The dynamic behavior of the whole vehicle model (with eAxle) will be validated by means of an optical measurement process or what so called (Object Tracking).

*Keywords: Electric axle drive system (eAxle); torque vectoring (TV); CO<sub>2</sub> emission reduction.*

## 1 Introduction

Nowadays, electric and hybrid vehicles are becoming increasingly important for many reasons, such as increasing oil prices and environmental pollution. Other contributory factors are legislation like, CO<sub>2</sub> emission reduction, taxes and zero emission zones. Nonetheless, e-mobility vehicles are much more efficient than conventional vehicles because they can regenerate, store and reuse kinetic energy.

The Schaeffler Group develops three types of eAxle: A high-voltage (HV) eAxle, which is very

suitable for electric and full-hybrid vehicles. This model has two electric motors (traction and TV). The second type is a 48-volt (48V) eAxle, which is optimally suited for mild- hybrid vehicles. It also has electric traction and TV motors. The third type is an (48V+TV) eAxle, which has just one 48V electric motor that is used in traction mode until a predefined speed limit and it then uses as a TV motor. This type has 3 planetary gear stages, two for traction mode and the third for TV mode. The last type of eAxle is a huge step forward in the development of the eAxle. A HV eAxle [Figure 1] is presented in this paper.

The HV eAxle has a traction motor with a maximum power and torque of 65 kW and 160 Nm respectively. With two planetary gear stages, the traction motor can drive the vehicle in full electric mode or assist the internal combustion engine (ICE).

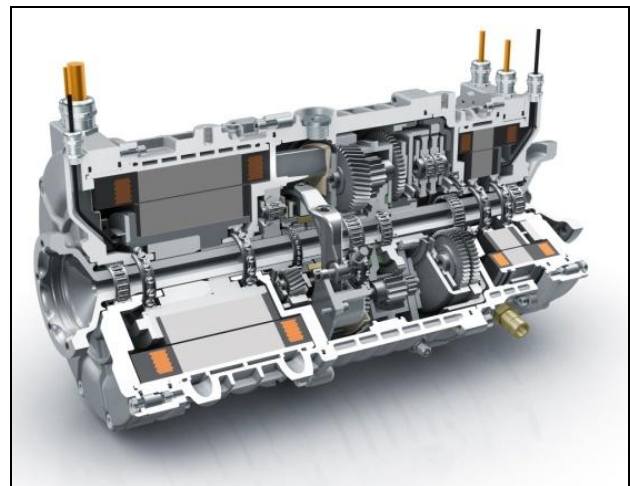


Figure 1: HV electric axle drive system

The TV motor has a maximum power of 6 kW, which is transmitted via a planetary transmission to supply up to 1100 Nm of delta torque between the left and right wheels.

The dynamics and agility of the vehicle can be significantly improved with the TV function. The steering characteristics are very responsive because

there is less vibration and the driver has an improved feel of the road conditions through steering feedback. Traction is improved especially with regard to critical road conditions like  $\mu$ -split. The most important benefit of the TV function is the improvement to the stability and safety without any adverse effects on the longitudinal dynamics and traction. This means no power losses or velocity decrease in comparison with other stability technologies that use a brake-split on different wheels.

An advantage of eAxle is the high potential for reducing fuel consumption and emissions because it is driven by electrical energy.

## 2 Modeling of the eAxle

The Schaeffler eAxle is a mechatronic system that contains mechanical parts (such as the transmission), electrical parts (electric motors) and thermal and control systems. It therefore requires a modeling approach that allows modeling of multi-domain systems. The modeling language Modelica® offers this facility and at the same time allows the model to be used on different simulation platforms. Therefore, the following model of the Schaeffler eAxle has been modeled using Modelica (www.modelica.org) in Dymola® which is a Modelica-platform tool.

### 2.1 Transmission model

The mechanics of the transmission [Figure 2] have been modeled with the 1D mechanics of the Modelica Standard Library (MSL). As an interface to the entire vehicle model, the transmission model has connectors for the torque vectoring motor, traction connector (which can be connected to the traction motor or ICE depending on the type of eAxle) and two output connectors (which are connected to the shafts of the left and right wheels). A deeper model level divides the model into the components "planetary gear sets", "differential" and "overlay gear set". The torsional rigidity of these components is determined by FEM analysis and mapped as linear springs in the model. The mass inertia is derived for all components from the CAD data and implemented as a rotating mass in the model.

The power losses in the traction path are implemented by means of a simulated map that has

been calculated using the WTplus® program. This map is just a 3D matrix that depends on the torque, angular velocity and temperature (which will be calculated in Abaqus and imported back into Dymola by co-simulation).

The traction gear ratios are 12.34 for first gear and 4.22 for second gear. The TV gear ratio is -37.72.

### 2.2 eAxle model

The transmission model is then integrated with the traction, TV motors and gearing signal into the eAxle model. The eAxle model has 3 input signals (throttle of the traction motor, which comes from the driver; throttle of the TV motor, which is regulated by the TV controller; and the required gear, which depends on the hybrid strategy).

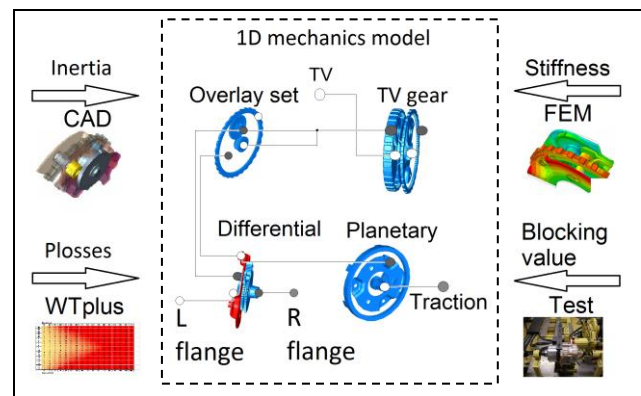


Figure 2: Transmission model

The efficiency-map and torque-over-angular velocity map of both motors (traction and TV) have been measured. The models of these motors have been implemented based on these maps (not a physical model)

The losses of the motors and transmission have been totalized to enable calculation of the actual temperature of the transmission and the temperature behavior on the eAxle housing by means of co-simulation with Abaqus.

The derating function of the electric motors has been defined by the motor supplier depending on thermal measurements of the motors. Thus, the derating function supplies a time-interval (for example, the traction motor is allowed to run with the maximum possible power for up to 5 seconds (overload time) and then has to run with less than or equal



the nominal power for at least 20 seconds (relief-time)). This function has been implemented with help of StateGraph library of MSL as shown in [Figure 3].

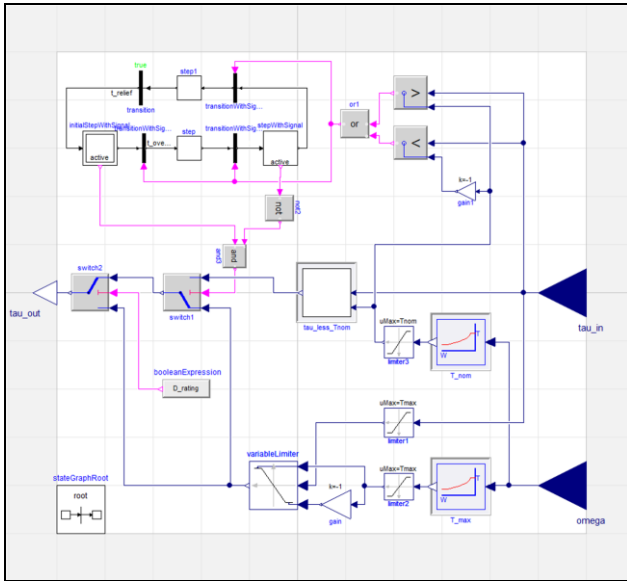


Figure 3: Derating model

### 2.3 TV controller

The TV controller is the unit responsible for defining the required torque of the TV motor in order to produce the necessary and effective yaw torque for the vehicle. [Figure 4] shows a diagram of the TV controller.

The TV controller calculates the desired yaw rate depending on the steering angle and some vehicle conditions as follows.

$$\psi_d^{\bullet} = f(\delta, v_x, a_y, \beta)$$

where,  $\psi_d^{\bullet}$  is the required yaw rate,  $\delta$  is the steering angle at the wheel,  $v_x$  is the longitudinal velocity,  $a_y$  is the lateral acceleration and  $\beta$  is the side slip angle.

The required yaw torque is calculated by comparing the required and the actual yaw rate.

$$T_{yaw\_d} = f(\psi_d^{\bullet}, \psi_{actual}^{\bullet})$$

Finally, the required vectoring torques that should be supplied by the TV motors are calculated in the following equations.

$$T_{V_{FA}} = \alpha \cdot \frac{b \cdot R}{-2 \cdot n_F} \cdot T_{yaw\_d}$$

$$T_{V_{RA}} = (1 - \alpha) \cdot \frac{b \cdot R}{-2 \cdot n_R} \cdot T_{yaw\_d}$$

where,  $T_{V_{FA}}, T_{V_{RA}}$ : Torque of TV motor of front/rear axle respectively,  $\alpha$ : Fraction ratio of TV between front/rear axle (if  $\alpha = 0$  only rear axle TV is active, if  $\alpha = 1$  only front axle TV is active, otherwise the TV of both axles is active with that fraction ratio),  $b$  is the vehicle track base,  $R$  is the wheel radius and  $n_F, n_R$  are the TV gear ratio of the front/rear axle respectively.

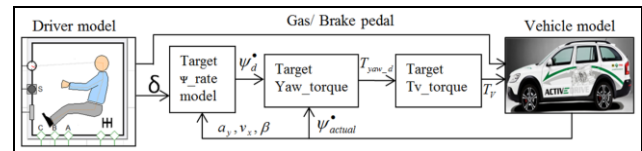


Figure 4: Diagram of TV controller

### 2.4 Vehicle model

The HV eAxle model was inserted into a drive train model and integrated together with a vehicle, driver and road models from the Vehicle Dynamics Library developed by Modelon®.

For the consumption analysis, the eAxle model was integrated with the rest of the vehicle model, driver and street models of the Power Train library developed by the DLR Institute of System Dynamics and Control.

### 2.5 Co-simulation between Dymola and Abaqus

A detailed thermal model of the eAxle has been modeled in Abaqus in order to carry out an adequate thermal analysis of the power losses in the eAxle due to the traction motor, TV motor and transmission. A co-simulation between Dymola and Abaqus is then used.

The Abaqus co-simulation technique can be used to solve complex systems that include electron-

ics such as control systems, electro-mechanics, hydraulics, heat transfer and pneumatics by coupling Abaqus with Dymola, a general-purpose logical modeling software distributed by Dassault Systems. Structural responses computed by Abaqus/Standard or Abaqus/Explicit are coupled at run time with solutions provided by Dymola [6].

The Abaqus modeling method called “physical modeling” is the complementary modeling method to “logical modeling”. Physical modeling is An FEM application, which uses a precise knowledge of the geometry to mimic the real world at a fine-grain level.

The Abaqus-Dymola coupled simulation capability targets simulations involving compliant response of a component or subassembly embedded in a larger, more complex system. All Abaqus features, including nonlinear materials, nonlinear geometric effects, and contact are available for use. In fact, any time-domain Abaqus model can be used as an Abaqus-Dymola co-simulation model with very few simple modifications[6].

The power losses of the eAxle in the Dymola model are defined with the same name as the actuators in the Abaqus model (Actuators are sources used to actuate the physical model). A virtual sensor is defined in the Abaqus model to measure the temperatures at some points and send them to Dymola. This sensor should have the same name as the input signal in the Dymola model. The Dymola model is then translated and the dymosim.dll file is selected by the plug-ins option in Abaqus.

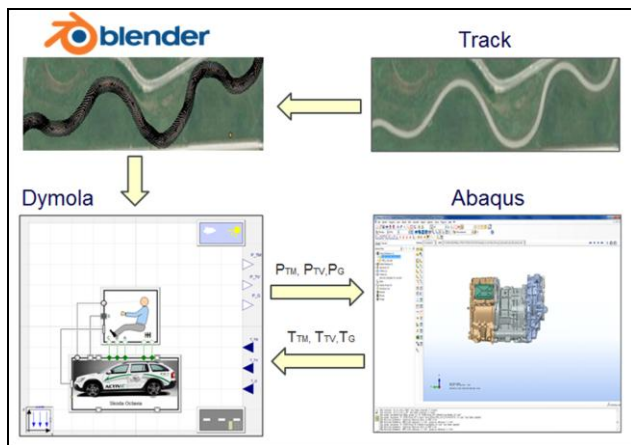


Figure 5: Co-simulation between Dymola and Abaqus, with a special realistic road.

In order to ensure a high level of actuation in the traction and TV motors, a special handling course road is used. The handling course road is imported as an image into Blender® in order to preprocess and generate the Modelica road file for implementation in the Dymola model [Figure 5]. The vehicle model is then simulated on the handling course road and the power losses of the eAxle are sent to Abaqus at every simulation step. Abaqus calculates the temperature values and sends them back to Dymola to run the next simulation step. These temperature values are necessary to calculate the power losses of the transmission.

### 3 Simulation Results

The consumption analysis and dynamic simulation of the vehicle have been calculated according to the vehicle data in Table 1.

Some comparisons have been made between the conventional vehicle model (without an eAxle) and the hybrid vehicle model (with an eAxle). The hybrid vehicle model is 80 kg heavier than the conventional vehicle model (weight of eAxle) for all following simulation cases.

Table 1: Vehicle specifications

Vehicle mass		2105 kg
Dynamic tire radius		0.3186 m
Frontal area		2.42 m <sup>2</sup>
Friction coefficient of road		0.9
Moments of inertia of the vehicle		I <sub>xx</sub> =709 I <sub>yy</sub> =3281 I <sub>zz</sub> =3358 kg.m <sup>2</sup>
Distance CG to the front, rear axle		1.269, 1.308 m
Vehicle track base		1.532 m
Mass of eAxle		80 kg
Traction motor	P <sub>max</sub>	65 kW
	T <sub>max</sub> , T <sub>nom</sub>	160, 140 Nm
	Speed max	14000 rpm
TV motor	P <sub>max</sub>	6 kW
	T <sub>max</sub>	30 Nm
	Speed max	14000 rpm
Battery	Energy	10 kWh
	V <sub>max</sub> , V <sub>min</sub>	365, 267 Volt
	SOC <sub>max</sub> , SOC <sub>min</sub>	0.8, 0.2

#### 3.1 Fuel consumption analysis

To determine the save fuel by using the eAxle model, the vehicle model is driven in the New European Driving Cycle (NEDC) for 100 km with four different cases: Conventional, hybrid with a ful-

ly charged battery, hybrid with a half charged battery and hybrid with a flat battery. The last case has been considered to specify the fuel saved in the case when the eAxle is used with no power in the battery, which means the eAxle can be driven only with the energy recuperated by braking. [Figure 6] shows the vehicle driving in two sequentially cycles of the NEDC. The eAxle firstly starts to recuperate the kinetic energy by braking and using it after that to drive the vehicle using electric power only for some period of time.

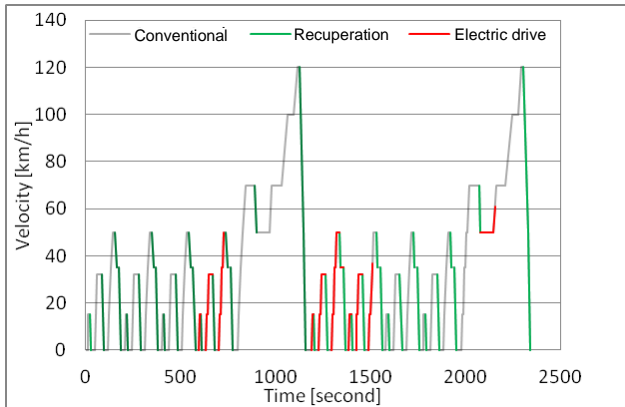


Figure 6: Hybrid vehicle driving in the NEDC with flat battery at beginning.

Although the eAxle makes the vehicle 80kg heavier, it allows a reduction in fuel consumption of at least 18% compared with a conventional drive. If the battery is pre-charged, the eAxle enables a reduction in fuel consumption of up to 38% depending on the state of charge of the battery.

The fuel consumption over time and the percentage of fuel saved by the hybrid vehicle in comparison with a conventional vehicle when driving up to 100 km are shown in [Figure 7].

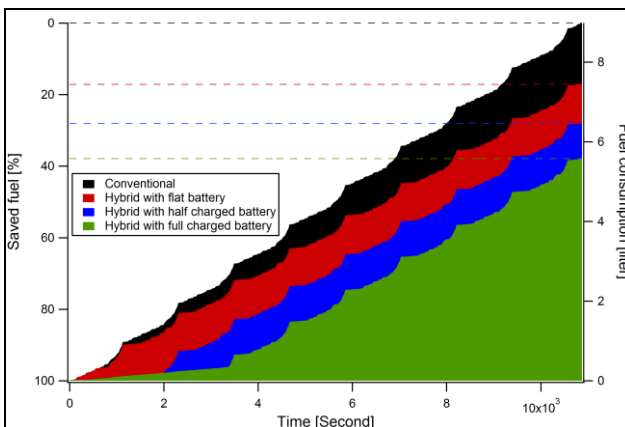


Figure 7: Fuel consumption in liters over time and percentage of overall fuel saving in a hybrid and conventional drive.

## 3.2 Dynamic analysis

### 3.2.1 Longitudinal dynamic

The eAxle almost makes a normal car into a sports car by using the 65kW traction motor to assist the ICE.

Simulation results show that the vehicle model without an eAxle takes about 6.6 seconds to accelerate from 0 to 100 km/h, while the hybrid vehicle took about 5.1 seconds, which means the eAxle can improve the longitudinal acceleration of the vehicle by 29% on average as shown in [Figure 8].

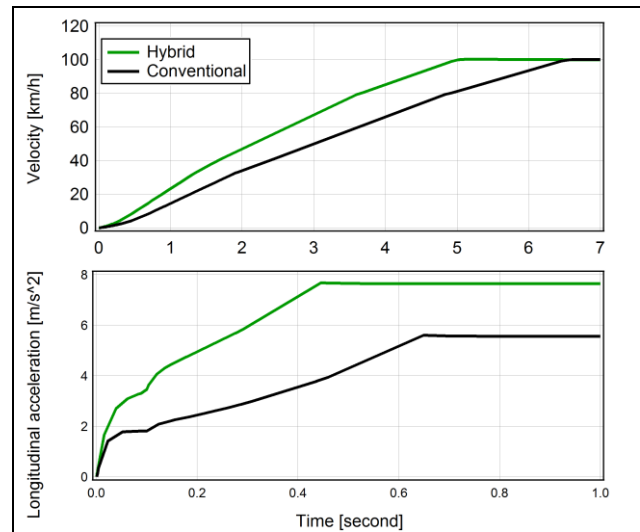


Figure 8: Acceleration from 0 to 100 km/h for both a conventional and a hybrid vehicle. i) Vehicle velocities. ii) Longitudinal acceleration up to the 1<sup>st</sup> second.

The eAxle can quickly supply a high traction torque even at low angular velocities, which gives the vehicle instantaneous high longitudinal acceleration. This fast response also gives the ICE a chance to supply a high torque in a shorter time because the eAxle drives the vehicle to a relatively high velocity which means the ICE runs with sufficient angular velocity to supply a high torque (ICE supplies low torques at low angular velocities). This explains the dramatic increase in longitudinal acceleration of the hybrid vehicle up to the 1<sup>st</sup> second in comparison with a conventional vehicle, as shown in [Figure 8].

### 3.2.2 Lateral dynamics

The TV feature has a big influence on the lateral dynamics performance because the TV motor with a maximum torque of 30 Nm can be used in conjunction with the TV gear to provide the vehicle

with a torque difference of up to 1100 Nm between the right and left wheels.

The vehicle is more stable and safer when it is driven in curves and can also be driven with a higher velocity. The driver will also notice the softer ride and ease of driving using the TV (driving pleasure).

Two simulation cases show the advantages of TV for the lateral dynamics, steering step response and cornering performance.

3.2.2.1 Steering step response

To verify the advantages of TV on the steering, the driver steers to 60 degree with 400 degree/sec while driving at 90 km/h. [Figure 9] shows the steering angle, while part b shows the yaw rate of the vehicle with and without TV. It is so clear that the yaw rate of the vehicle has a smaller overshoot when using the TV because it increases the torque on the outer wheel and decreases torque on the inner wheel. The yaw rate also has less fluctuation and reaches a steady state faster because the TV compensates the fluctuation by biasing the difference in torque between the wheels.

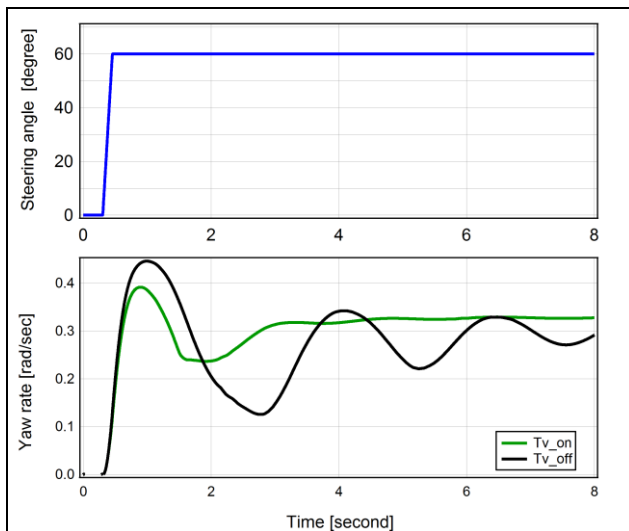


Figure 9: Step response test of a vehicle driven at 90 km/h. i) Steering angle. ii) Yaw rate with and without TV.

3.2.2.2 Cornering performance

A circular road with a 40 m radius curvature has been used to simulate the vehicle model with and without the TV function. The vehicle is driven at 65 km/h.

The simulation results [Figure 10] show that:

- TV decreases the gradient of the steering angle ( $\delta$ ) over the lateral acceleration of the vehicle [region1]. This means the steady-

state behavior of the cornering performance of the vehicle is improved by using TV.

- The maximum level of lateral acceleration can be extended by using TV [region2], which gives the vehicle a wider range of stability.
- The TV decreases the necessary steering angle at high lateral acceleration, which is reflected in increased safety and driving pleasure.

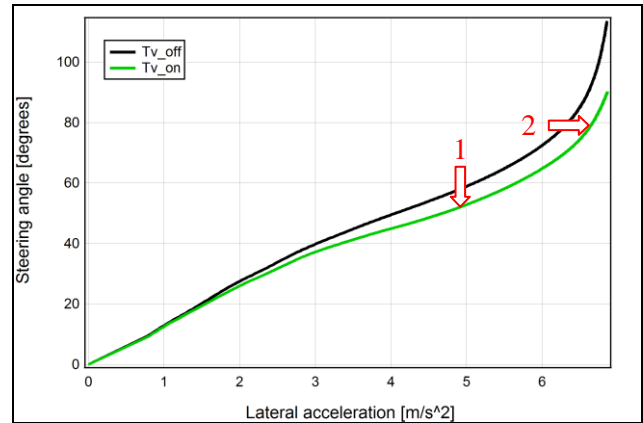


Figure 10: Steering angle over lateral acceleration with a 40 m radius curvature at 65 km/h.

The vehicle takes 8.2 seconds to negotiate the curve using TV and 8.3 seconds without TV. In addition, the total energy used by the traction and TV motor to negotiate the curve with TV is 1% less than the energy required when the TV is switched off (traction motor only) as shown in [Figure 11].

A reduction in fuel consumption of approximately 1.9% could be achieved by using the ICE instead of the traction motor of the eAxle (This configuration is suitable for special designs of low-voltage eAxle). This means the TV feature not only improves the lateral dynamic performance, but also saves some fuel.

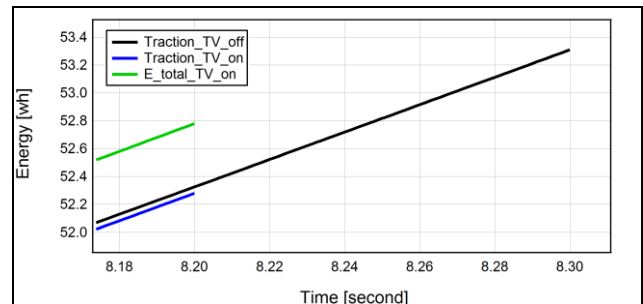


Figure 11: Energy consumption during cornering with and without TV.

### 3.3 Dymola-Abaqus co-simulation

The vehicle is driven at 60 km/h on the handling course road and Abaqus calculates the temperatures on the housing of eAxle depending on the power losses [Figure 12].

These temperatures are used to define the conditions of the electric or hybrid strategy.

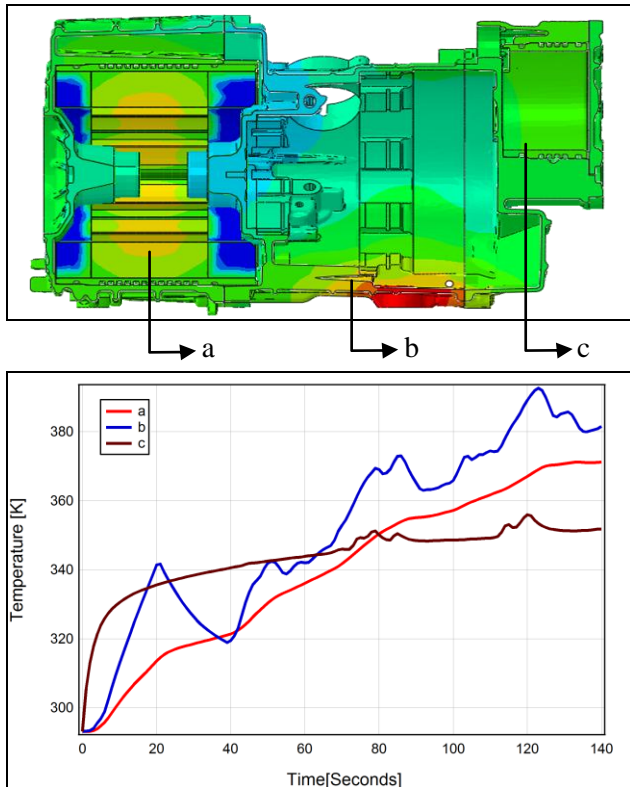
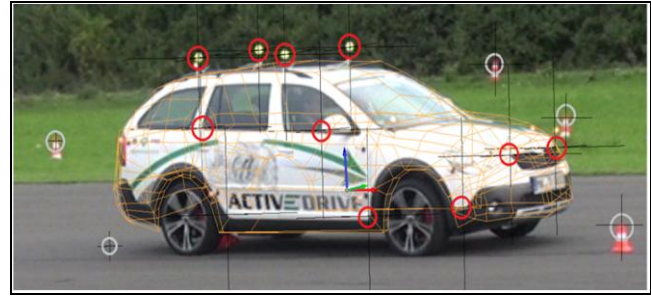


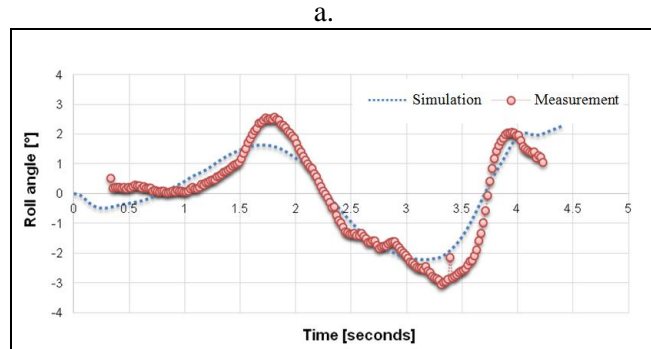
Figure 12: Temperatures on the eAxle housing a) on the traction motor. b) on the transmission and c) on the TV motor.

### 3.4 Validation of vehicle model

The entire vehicle model was validated on the basis of the lane-change test (ISO standard 3888-2) by using an optical measurement process. The measurement process analyzes the motion of optical markings [Figure 13.a] in a video recording of the driving maneuver (object tracking). With the corresponding software Blender®, the variation in vehicle position and orientation on the test track can be determined over time[3]. A comparison between roll angle of the test vehicle by the object tracking and simulation result is presented in [Figure 13.b].



○ Inertial system. ○ Vehicle



b.

Figure 13: a) Test vehicle with tracking markers. b) Roll angles in simulation and object-tracking validation.

## 4 Conclusions and Outlook

The modeling of a multi-domain system in Modelica, and the concept of building an electric active drive system for reducing fuel consumption , improving traction and providing yaw stability control have been discussed.

Modelica has a major advantage for dealing with a model that has many components from different disciplines.

Simulation results, which prove the advantages of the eAxle system in terms of dynamics and consumption are presented. A co-simulation has been developed between Dymola and Abaqus to investigate the thermal effects on the eAxle.

A realistic road has been constructed and preprocessed with other animation tools and can be easily and flexibly exported to Dymola .

Another simulation case has been started to investigate the effect of recuperation and the extra weight of the eAxle on the fuel consumption by considering the influence of elevation when the vehicle is driven on a road with a gradient. Unfortunately, the model of the driver in the PowerTrain library is unable to maintain the vehicle's velocity close to the

requirement when the vehicle is driven downhill, which leads to non comparable consumption results.

Vehicle object tracking for validation the vehicle performance has been presented. A validated case between simulation result and reconstructed data from object tracking has been compared and shown.

## References

- [1] Smetana, „Schaeffler E-Axle Drive“.
- [2] „Active electric differential for future drive trains“. Dr. Tomas Smetana, Thorsten Biermann, Prof.-Dr. Bernd-Robert Höhn, FZG München, Franz Kurth, FZG München and Dr. Christian Wirth, FZG München.
- [3] „Lastdatenermittlung in Elektrischen Achsantrieben und Betriebsfestigkeitsbewertung für das Subsystem Elektrische-Maschine“. D. Knetsch, F. Rettig, Dr. M. Funk, H. Awad, Dr. T. Smetana, M. Gramann. 40.Tagung des DVM-Arbeitskreises Betriebsfestigkeit.
- [4] „Das aktive eDifferenzial für den Antrieb der Zukunft“. 13. ITI-Symposium. Frank Rettig, Dr. Tomas Smetana and Roman Sauer.
- [5] „Interfacing Abaqus with Dymola: A High Fidelity Anti-Lock Brake System Simulation“. Modelon AB , 7<sup>th</sup> Modelica Conference 2009.
- [6] „Logical-Physical Modeling Using Abaqus and Dymola“. Dassault Systems, 2012
- [7] Modelica® - A Unified Object-Oriented Language for Systems Modeling. Language Specification. Version 3.3.
- [8] Modelica®, [http://www. Modelica.com](http://www.Modelica.com).
- [9] Abaqus®, Dassault systems <http://www.3ds.com>.
- [10] Blender 3D. <http://www.blender.org>.

# Utilizing Object-Oriented Modeling Techniques for Composition of Operational Strategies for Electrified Vehicles

Sebastian Hämmerle  
Vorarlberg Univ. of Appl. Sc.  
Austria

[Sebastian.Haemmerle@students.fhv.at](mailto:Sebastian.Haemmerle@students.fhv.at)

Marco Kessler  
Modelon GmbH  
Germany

[Marco.Kessler@modelon.com](mailto:Marco.Kessler@modelon.com)

Markus Andres  
Modelon GmbH  
Germany

[Markus.Andres@modelon.com](mailto:Markus.Andres@modelon.com)

## Abstract

The paper introduces a new concept of modeling the overall control unit of hybrid electric vehicles in *Modelica*. The work focuses on a structure which can simulate substantially different vehicle concepts without changing the structure of the control unit. Based on this universal implementation different scenarios can be simulated rapidly and consequently cheaply, including fundamentally different drive trains ranging from conventional to purely electrical including hybrid versions.

*Keywords:* object-oriented control unit modeling; control unit template modeling; object-oriented electrified vehicle modeling

## 1 Introduction

Ever increasing demands on fuel efficiency are requested from both politicians and customers. This should happen without increasing the costs and without compromising on performance of the vehicle [2]. Therefore, simulation in the concept stage of a vehicle development is getting more and more important, because with the aid of simulation design decisions can be supported in a more efficient overall process that is consequently resulting in cost savings [4]. This is especially true when designing complex systems which are state-of-the-art in the form of hybrid electric vehicles (HEV).

Nowadays, several different HEV topologies are state-of-the-art, for example series hybrid, parallel hybrid or power split hybrid. Furthermore, from every topology multiple modification levels are available. Until now, if a rough layout of a HEV has to be simulated, a very specific CU (control unit), specially adapted to each different vehicle topology has been modeled. This process is error-prone, time consuming and in turn associated with high costs. Due

to these facts the manufacturers postulate the development of a universal CU for simulations.

Implementations of individual HEV topologies have been discussed many times in literature, but the implementation of a standardized template for a universal CU has not yet been provided.

Examples are given in [8] and in [9], where fuel consumption simulations - on basis of two commercial *Modelica* libraries: the *PowerTrain* library [5], [7] and the *SmartElectricDrives* library [1] - of different HEVs are discussed.

Another example of a library for simulation of HEVs is given in [3]. The used library is called *eVehicleLib* and can be used for fuel and energy consumption simulations as well as other purposes.

Both, the *PowerTrain*- and the *eVehicleLib* library contain models for process control, i.e. for the operating strategy but no template of a universal CU for different vehicle topologies. The concept presented in the paper fulfills this request of the manufacturers.

The paper focuses on HEV structures, what is reasonable as purely electric or conventional drive trains may be treated as a simplification of suitable hybrid versions. An example will be demonstrated in Section 5 where the hybrid car (BMW i8) is simulated as a conventional vehicle by just changing controller parameters.

## 2 Hybrid Electric Vehicles

Basically, HEVs can be classified on either the type of the power train or by their degree of hybridization. Speaking of type of power train, one can distinguish between series-, parallel- and power-split hybrids. The classification by degree of hybridization contains micro, mild and full hybrid.

Due to the fact that a HEV, in contrast with a conventional vehicle, has two energy converting systems -

usually a combustion engine (CE) and an electric machine (EM) - more degrees of freedom in generating traction energy are available. Which possibilities are available depends on the arrangement of the CE and the EM in the drive line, i.e. on the HEV topology [2].

### 3 Modeling Approach

To design a template of a universal CU, which can be used to simulate different kinds of HEVs, it must first be understood how the CU interacts with the remainder of the entire vehicle system. Therefore, a decomposition of an entire vehicle system into basic objects is necessary. The first division is done into four main parts: a drive environment, an environment, a vehicle itself and the control unit.

The drive environment emulates the human driver, i.e. it influences the vehicle in longitudinal direction with the accelerator and brake pedal (lateral direction is neglected). The schematic function of the drive environment is illustrated in Figure 1. Depending on a reference velocity (supplied by the driving cycle) and an actual velocity (supplied by the vehicle) the accelerator- and brake pedal position is computed by the driver. These signals are sent to the CU which computes signals for the vehicle. Therefore, a direct interaction between the CU and the drive environment is given.

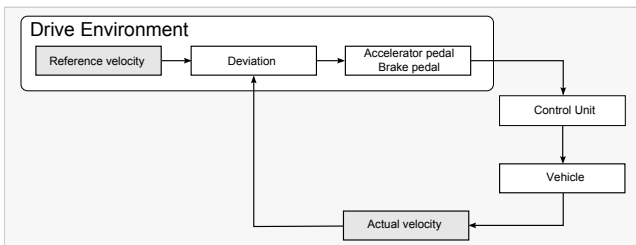


Figure 1: Schematic structure of the drive environment

In the vehicle it is distinguished between objects which are required for generating traction energy and the ones which are not, i.e. the vehicle is grouped into a drive line and auxiliary units, see in Figure 2. The CU provides on the one hand signals for the drive line, e.g. torque requirements for the CE and the EMs and on the other hand for the auxiliary units, e.g. the available power for the auxiliary units. Furthermore, sensor signals from the drive line are necessary in the CU, e.g. the actual vehicle speed. Therefore, the CU has a direct interaction between the objects from the vehicle.

The environment includes all effects from outside of the entire vehicle system and provides it to the ve-

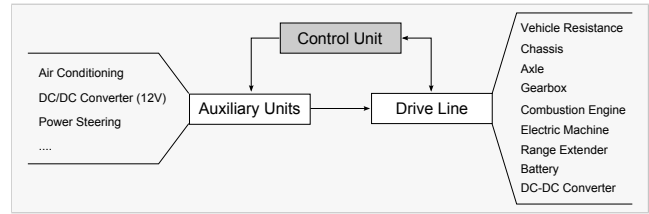


Figure 2: Schematic structure of the Vehicle

hicle, see Figure 3. The CU requires signals from the environment which as well influence the vehicle and therefore, an indirect interaction between the CU and the environment exists.

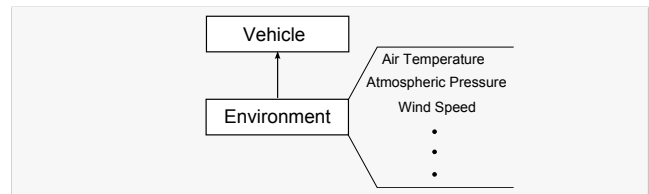


Figure 3: Schematic structure of the environment

Figure 4 shows the decomposition of the entire system in summarized form, whereas the interaction between the objects is indicated by arrows.

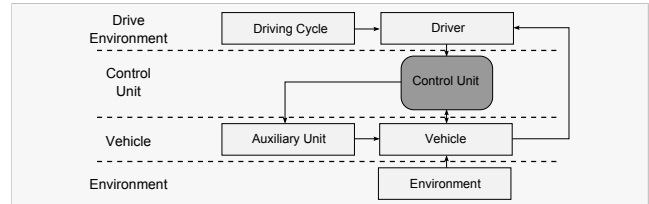


Figure 4: Schematic structure of an entire system

The interaction between the CU and the rest of the entire vehicle system is clearly defined. Now, the decomposition of the CU has to be carried out in a way that it can be reused for different HEV topologies. To make the structure flexible enough a controller with two levels is introduced (see Figure 5). In the primary layer the HCU (hybrid control unit) determines the current driving mode. Depending on the output of the primary layer the second level of hierarchy computes the desired signal for the corresponding drive line component (see Figure 5).

- BCU (brake control unit)
- ESCU (electric storage control unit)
- CCU (clutch control unit)
- GBCU (gearbox control unit)



- ECU (engine control unit)
- EDCU (electric drive control unit)

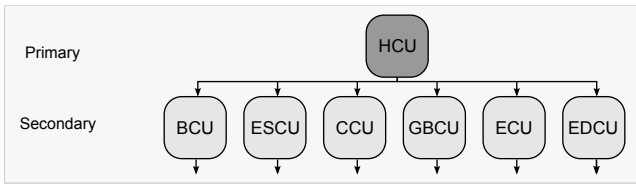


Figure 5: Schematic structure of the CU

To test the controller three parts are necessary, of which the first is to design a template for the overall system, which is shown in Figure 6. For this purpose a library with basic components was created to replace the corresponding parts of the template. As the focus of this paper is to present the structure of the controller the models for testing (*Drive Environment*, *Vehicle* and *Environment* in Figure 6) are not discussed here in more detail. Moreover a considerable amount of attention was paid to the fact that well defined interfaces of each model were used in the template and therefore every component model can be adapted or changed easily, thanks to Modelica’s object-orientation. In the following section we will focus on the implementation of the universal control structure that is shown in the *Control Blocks* part of Figure 6.

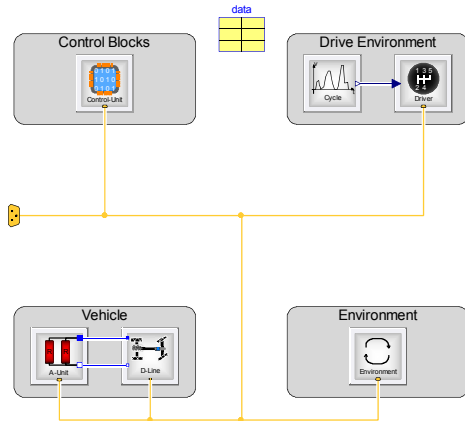


Figure 6: Template of the entire system

## 4 Universal Control Unit Modeling

Manufacturers tend to protect the control strategy as trade-secret because it represents one of their core knowledge. Therefore, the implementation of each part of the CU must be created in a way that it can easily be adapted and/or replaced. The structure of the

implementation of the CU is based on Figure 5 and its implementation is shown in Figure 7.

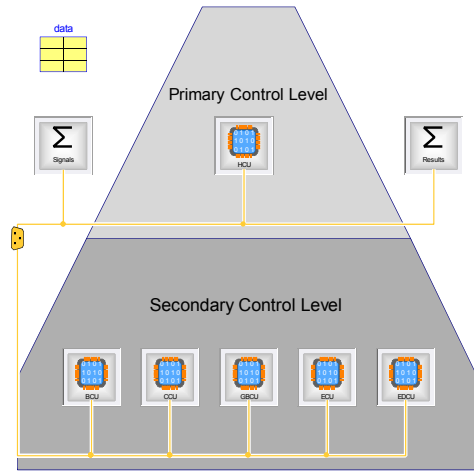


Figure 7: Template of the universal CU

### 4.1 Bus Structure

Looking at Figure 7 one can easily see, that the implementation is heavily relying on a bus structure. It was applied to the massive amount of control signals within the controller that have to be dealt with. Moreover it enhances the flexibility and maintainability of the overall controller. To reduce errors within connecting signals from the bus and as well enhancing maintainability connectors are introduced that only feed values from the bus to a signal connector of suitable type (Real, Integer, Boolean). Although those adapter models are very simple they are valuable in certain cases. When e.g. a variable on the bus is renamed only the connector has to be modified in contrast to every single connection relying on that variable.

In the current implementation the bus has a flat structure without any grouping of variables. After it turned out that the bus has close to 100 variables it would be well worth the effort to restructure the bus with a hierarchy related to e.g. Figure 4.

### 4.2 Primary Control Level

The primary control level solely consists of the HCU as shown in Figure 7. It is the centerpiece of the CU as it supervises all the other control units of the secondary control level. To implement a universal HCU it is necessary to consider all possible driving modes of all HEV topologies. These are

1. Standstill with CE off (*StandStill*)

2. Standstill with CE on (StandStillEngine)
3. Electric Driving (ElectricDriving)
4. Friction braking (Braking)
5. Regenerative braking (RegenerativeBraking)
6. Boosting (Boosting)
7. Shifting of the Operation Point (Shifting-OperationPoint)
8. Conventional Driving (ConventionalDriving)

whereas the names in the brackets denote the variable names used in the model itself.

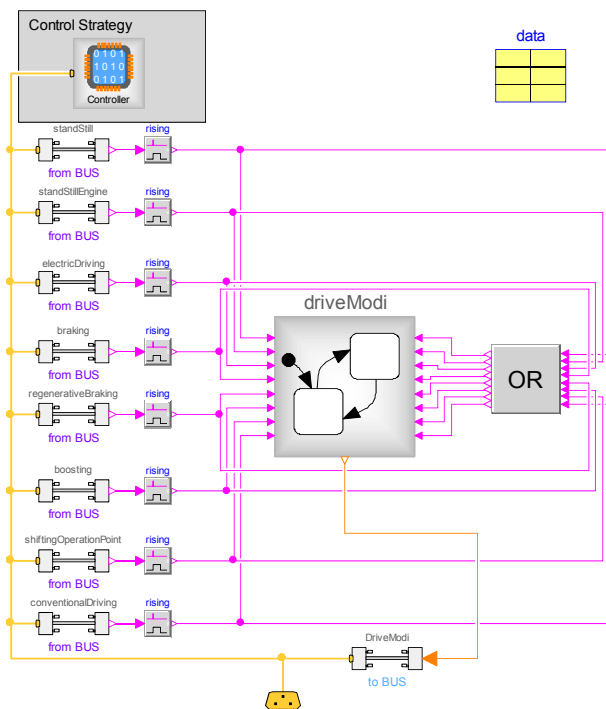


Figure 8: Implementation of the HCU

The HCU (Figure 8) has to decide which of those driving modes is active e.g. depending on driver inputs or internal states of the vehicle. The first prerequisite to be able to take that decision is to know which of those driving modes are available in the current vehicle configuration. Therefore parameters to activate driving modes independently of each other are provided in the data record shown in Figure 8.

In the current implementation independent models are used to decide which driving mode is active as shown in Figure 9. All of them have Boolean outputs each one representing a possible driving mode. This implementation guarantees maximum flexibility but is not the most straight-forward implementation.

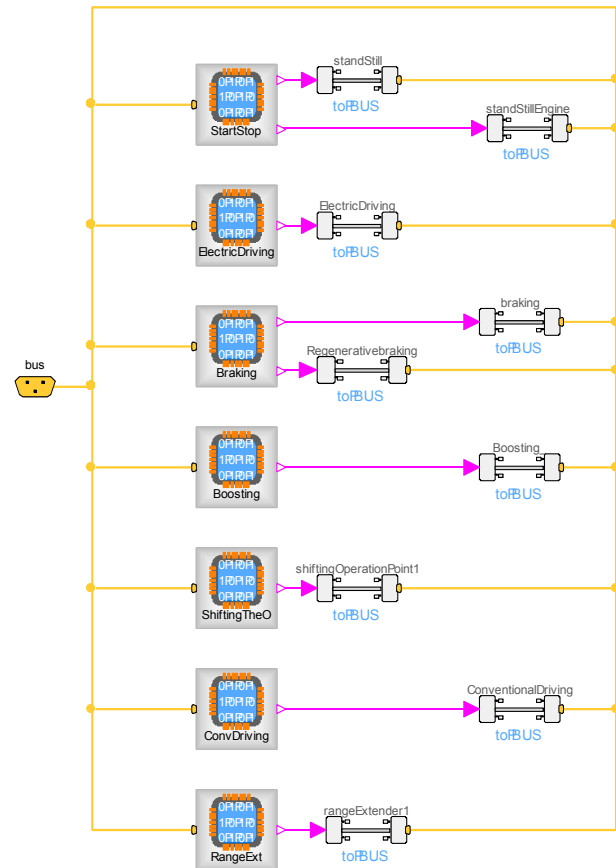


Figure 9: Implementation of the Control Strategy within the HCU

To have distinct behavior it has to be ensured that in each time step only one driving mode is active within the HCU. To ensure that, the modeling is done via the *state graph 2* library<sup>1</sup>. The implementation shown in Figure 10 is a combination of eight parallel loops, one for each driving mode. This model is used named *driveModi* in Figure 8. The loops consist of an activate and a deactivate transition and therefore of an on and an off step. The output of the HCU is an integer variable that is put to the bus indicating the current drive mode. This drive mode is then utilized by the secondary level controllers (Figure 7) to generate input signals for the drive train components. As a result of this structure it is not only possible to simulate HEVs of different structure, but also purely electric vehicles and conventional vehicles. Therefore, the modeler can simply choose a hybrid vehicle topology and based on that the possible and desired driving modes are enabled in the HCU.

The range extender operation is an exception to that rule. This mode can be interpreted using a source of power for the battery without any power directly

<sup>1</sup>For further information it is referred to [6]

transferred to the power train. Alternatively it could be seen as part of the auxiliary units as it does not generate traction force. It can therefore not be integrated in the integer variable that represents the driving mode, as some other driving mode has to be active to compute the control signals for the components of the drive train. Then the integer value for the driving mode would have to take two values. It is therefore represented by an additional Boolean variable on the bus.

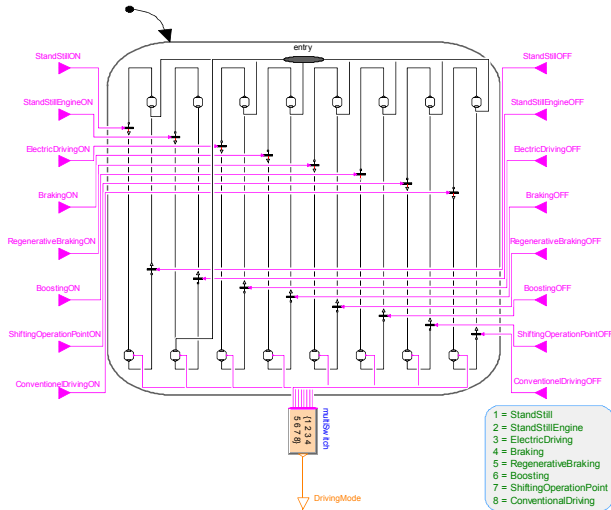


Figure 10: Driving mode selection via state graphs

Consequently the driving modes are represented by an integer variable (driving mode 1 - 8), which is computed via the state graphs and a Boolean variable (driving mode 9), which is computed in the Controller block.

**Modifications**

The driving modes are simple to adapt or to replace as only the Boolean output has to be fed with a meaningful value. If an additional driving mode should be implemented a new driving mode controller in the *Controller* block and an additional loop in the *DriveModes* block has to be implemented.

**4.3 Secondary Control Level**

In the following sections the second layer controllers are shown starting with the engine control unit that is discussed in a bit more detail whereas the others shall be described only briefly.

**4.3.1 Engine Control Unit**

The ECU is responsible to calculate the normalized torque request of the CE. Furthermore, the signals for the starter generator and the signals to bring the CE into idle mode or switch it off are computed in this controller. The main areas of functionality are shown on the left side of Figure 11. These are

- Conventional Driving
- Shifting the Operating Point and
- Boosting

represented by different paths through the model within the left three boxes in the model.

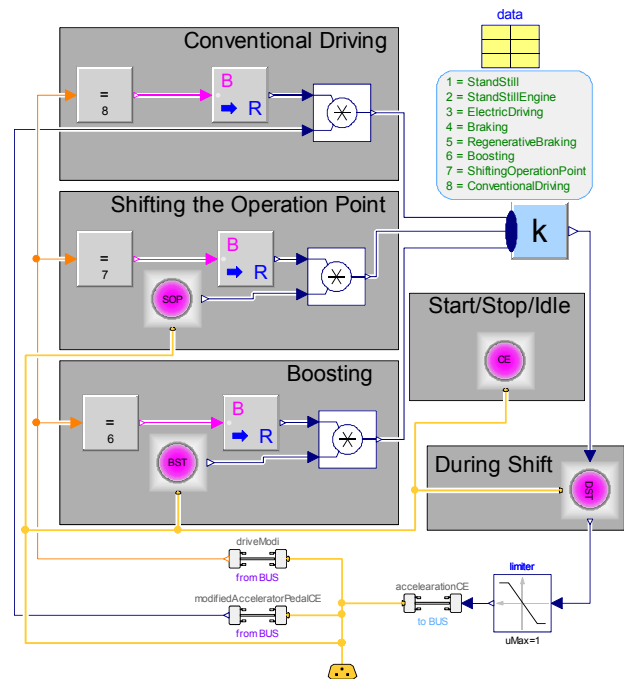


Figure 11: One possible controller of the combustion engine

They compute an output signal that represent the combustion engine's torque demand depending on the variable *driveModi* that is taken from the bus via the adapter shown in the lower part of Figure 11. The combination of compare block, Boolean to real converter and multiplication with the output basically act as an activation and deactivation of the single path. The *SOP* and *BST* blocks represent implementations of the functionality indicated by their name.

The additional blocks for *Start/Stop/Idle* and *During Shift* take responsibility that the CE acts like one would expect during the operation states indicated by their name.

Flexibility is granted by the possibility to replace the blocks for *SOP*, *BST* etc. or by replacing the overall *ECU* (Engine Control Unit) within the *CU*. Which possibility is more suitable has to be decided within the application that is targeted.

### 4.3.2 E-Drive Control Unit

The EDCU is responsible to calculate the normalized torque requests of the EMs. Furthermore, in this controller it is ensured that the computed signals are sent to the corresponding EM (in case more than one EM is available).

The implementation of the EDCU is split into different areas, i.e. one for each driving mode where an EM is involved. By this structure it is clearly defined how the normalized torque request signal for the EM is calculated in every driving mode. Moreover, by this implementation there is a clear separation enabling simple replacement of strategies depending on driving modes.

Due to the bigger amount of driving modes that the electric drives are included compared to the combustion engine the model gets more complex than the one in Figure 11 as one can see from Figure 12.

### 4.3.3 Gearbox Control Unit

The task of the GBCU is to evaluate the moment of shifting a gear up or down. In a real vehicle the shifting is dependent on different parameters, e.g. vehicle speed, slope of the road, actual gear level. To simplify the implementation it is assumed that the shifting is only dependent on the actual vehicle speed, i.e. the gear shift depends on predefined threshold values. The output is the actual gear level and is represented by an integer variable.

### 4.3.4 Clutch Control Unit

The CCU is responsible for opening and closing the clutches. The implementation is similar to the EDCU and ECU. That means that the CCU is split into different areas and, therefore, it is easy to adapt, expand or replace the controller.

### 4.3.5 Brake Control Unit

The BCU has the task to split the required brake torque into a recuperation brake and a friction brake amount. Two different regenerative braking strategies are im-

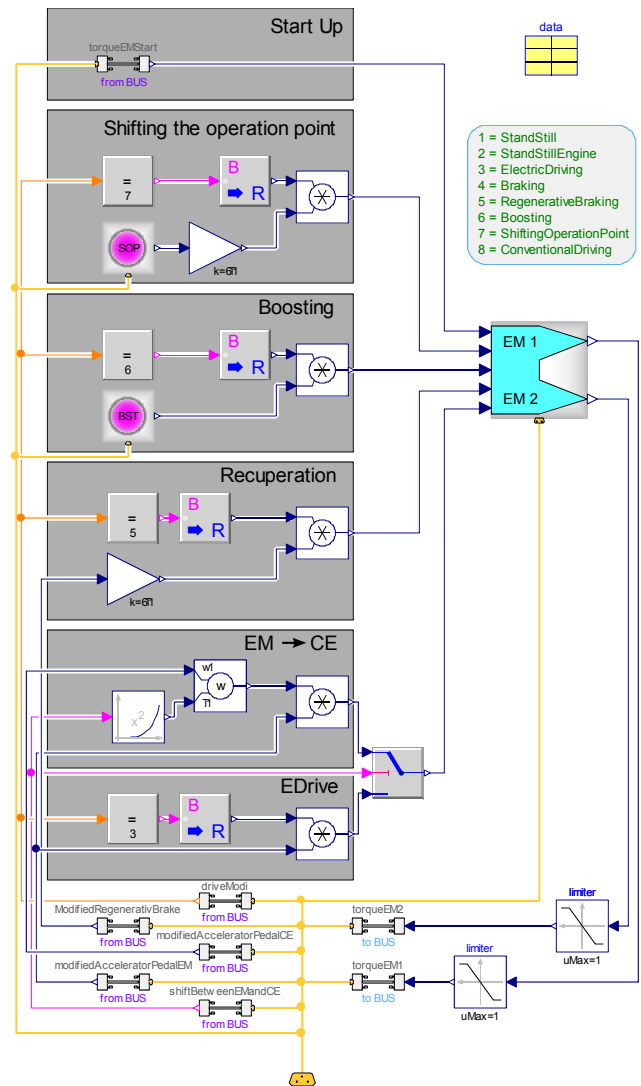


Figure 12: One possible controller of the electric machine

plemented: the series-<sup>2</sup> and the parallel<sup>3</sup> recuperation strategy.

## 4.4 Other Models

In Figure 7 models besides the primary and secondary control level. These will be discussed in this section.

### 4.4.1 Signals

For simulation of a conventional vehicle the accelerator pedal position can be directly used to scale the

<sup>2</sup>As long as the required deceleration is not higher than the provided generator deceleration, the generator brake is solely used. When the required deceleration increases, the friction brake is added.

<sup>3</sup>The generator and the friction brake are acting together in a fixed ratio.

maximum torque of the CE. In comparison to that, HEVs have at least two energy converting systems. Therefore the accelerator pedal position must be used to scale the maximum available torque for forward motion, which in most cases results from a combination of the torques of the CE and the EM.

In this block the conversion of the accelerator- and brake pedal position into normalized input signals is implemented.

#### 4.4.2 Results

The *Results* block is for the computation or display of values which are necessary for evaluating the simulation, for example the fuel consumption.

### 4.5 Auxiliary Units

The auxiliary units are all power units, which are not primarily needed for generating traction energy. Therefore they are not part of the HCU directly as it focuses on components for traction energy generation. Still in modern vehicles a multitude of auxiliary units can be found, e.g. an air conditioning system or a lube oil pump. The power controlling of the auxiliary units is realized via a priority list, i.e. the available electric power is shared depending on the driving mode.

Figure 13 illustrates the schematic function of the priority list. The required power from the drive line and from the auxiliary units is sent to the priority list block. This block splits the available power depending on the drive mode and sends the results to each block.

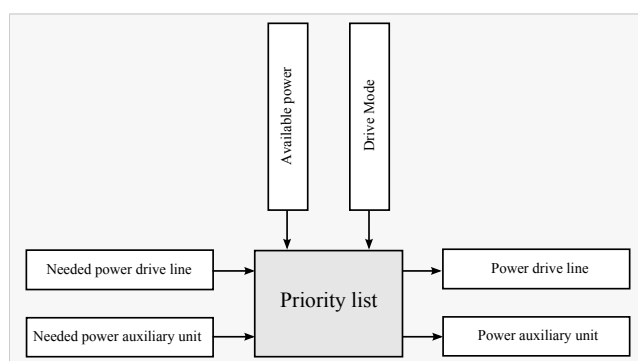


Figure 13: Schematic function of the priority list

## 5 Simulation

To demonstrate the universal applicability of the designed controller, two different vehicles are simulated. One target of simulations like these could be to judge

their fuel consumption during common drive cycles. The intention was to pick one vehicle with very high complexity, which is why the BMW i8<sup>4</sup> was chosen. Additionally the BMW i8 was operated as it is intended to and as a conventional vehicle by deactivating the hybrid operation modes. The BMW i3<sup>5</sup> was picked due to its very different structure that has to be covered with the same controller.

The driving modes in shown the simulation results in Figures 15, 16 and 18 are referring to the ones listed in Section 4.2.

It is important, that not all parameters for the models were exactly known which make the absolute results differ from real world values. Still relative comparisons based on different operating strategies are feasible and demonstrate the flexibility of the designed controller.

### 5.1 BMW i8

It is exemplarily shown how a comparison between the fuel consumption of the BMW i8, used as a conventional vehicle, and the BMW i8, used as an HEV could be made (driving cycle: NEDC).

At first a fuel consumption simulation of the BMW i8 as a conventional vehicle is run. For that solely the conventional driving mode is enabled in the HCU, i.e. only stand still with engine on, friction braking, and conventional driving are possible.

The second step is to run the fuel consumption simulation of the BMW i8 as an HEV. Therefore, the suitable driving modes are enable in the HCU, i.e. start-stop automatic, regenerative braking, electric driving, boosting, shifting of the operation point and conventional driving.

The changes between the simulations come down to selecting the boolean parameters that enable or disable the driving modes mentioned above within the parameter window. No modifications of the single controllers in any level of hierarchy is necessary.

### Result

The results of the fuel consumption simulations utilizing the NEDC are shown in Figure 14, whereby the dashed line represents the BMW i8 as a conventional vehicle and the constant line in the BMW i8 as a HEV. More important regarding this paper is how the controller behaves with respect to the driving modes that

<sup>4</sup>Is a special case of a parallel structure, called through the road hybrid or axle split hybrid.

<sup>5</sup>Is available as a purely electric vehicle or as series HEV

are activated. This is presented in Figure 15 and 16. In Figure 15 mainly the modes 1 (StandStill), 3 (Electric-Driving) and 5 (RegenerativeBraking) are active during the first 800 seconds of simulation. Afterwards the highway cycle starts, where 7 (ShiftingOperation-Point), and 8 (ConventionalDriving) are active.

If the BMW i8 is configured by a few clicks to be operated as a conventional vehicle only the modes 2, 4 and 8 are available resulting in Figure 16.

The fuel consumption of the BMW i8 as a conventional vehicle is  $0.98 \frac{1}{\text{NEDC}}$  ( $8.89 \frac{1}{100\text{km}}$ ) and as an HEV  $0.61 \frac{1}{\text{NEDC}}$  ( $5.53 \frac{1}{100\text{km}}$ ). In other words, the use of the BMW i8 as an HEV instead as a conventional vehicle effects a fuel saving of approximately 38 %.

### BMW i3

This investigation focuses on the effect of a range extender (10l tank volume) on the driving range of the BMW i3 (driving cycle: WLTP).

At first a driving range simulation with the BMW i3, used as a purely electric vehicle, is carried out. Therefore, the available driving modes are enabled in the HCU, i.e. start-stop automatic, electric driving and regenerative braking. Afterwards, the second drive range simulation is executed, with the difference that the range extender mode is additionally enabled in the HCU.

### Result

The results of the driving range simulations are illustrated in Figure 17. The upper curve is driving range of the BMW i3 as a purely electric vehicle and the lower curve the driving range of the BMW i8 with an additional range extender.

The driving range of the purely electric vehicle is 112.4 km and with the additional range extender 287.7 km. This links to a surplus of driving range of approximately 175 km.

Figure 18 shows the driving modes that are active during the simulation of a WLTP cycle for the purely electrically driven BMW i3 without range extender. It shows that only the modes 1 (StandStill), 3 (Electric-Driving) and 5 (RegenerativeBraking) are active in this case. The controller structure was the same as for the simulations carried out with the BMW i8, with necessary changes to the secondary level controllers due to other components used.

## 6 Conclusion

To summarize, the presented controller structure can be seen as a good starting basis for manufacturers for the simulation in the early design phase of a vehicle propulsion system. The presented implementation of a universal control unit provides a simple and time saving possibility to quickly simulate fundamental design changes in the system, which was the target for this development. Additionally the unification of the controller structure comes with other advantages like quicker orientation in non-familiar projects.

The presented implementation is in a prototype stadium and there are several parts that could be enhanced. These are e.g. the integration of the range extender mode that could be added to the auxiliary components instead of the driving modes. Another thing that should be reviewed thoroughly is the mixture of boolean and integer variables that are determine the driving mode. The current implementation goes for maximum flexibility but most likely more intuitive possibilities exist to solve this problem.

## References

- [1] J. V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, and F. Pirker. The SmartElectric-Drives Library - Powerful Models for Fast Simulations of Electric Drives. *The Modelica Association*, 571-577, 2006.
- [2] P. Hofmann. *Hybridfahrzeuge: Ein alternatives Antriebskonzept für die Zukunft*. Springer, 2010.
- [3] S. Kutter and B. Bäker. eVehicleLib - Eine Modelica-Bibliothek zur Simulation von Fahrzeugen mit alternativen Antrieben. *Symposium Simulationstechnik - ASIM*, 2009.
- [4] Q. Li. *Developing Modeling And Simulation Methodology For Virtual Prototype Power Supply System*. PhD thesis, Virginia Polytechnic Institute and State University, 1999.
- [5] M. Otter, M. Dempsey, and C. Schlegel. Package PowerTrain. A Modelica library for modeling and simulation of vehicle power trains. *The Modelica Association*, 23-32, 2000.
- [6] M. Otter and I. Dressler. Stategraph - A Modelica Library for Hierarchical State Machines. *The Modelica Association*, 569-578, 2005.

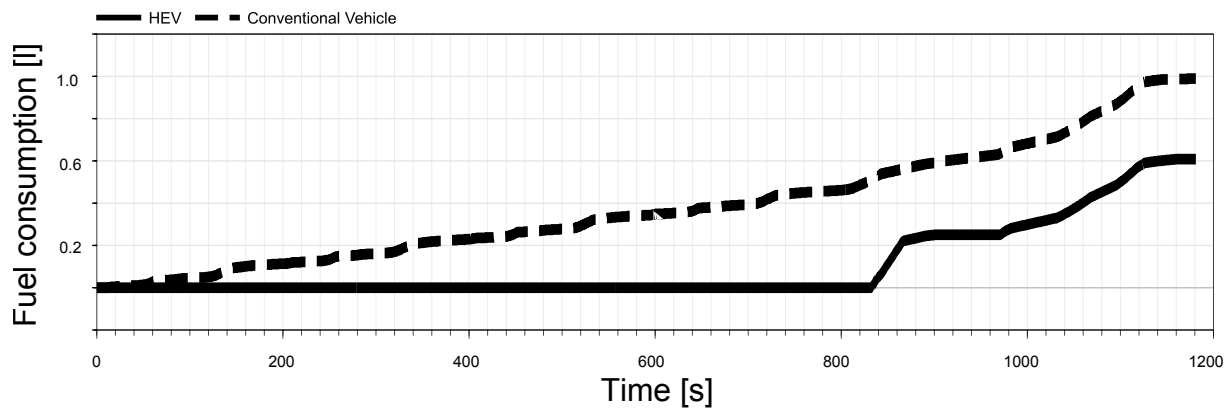


Figure 14: Fuel consumption

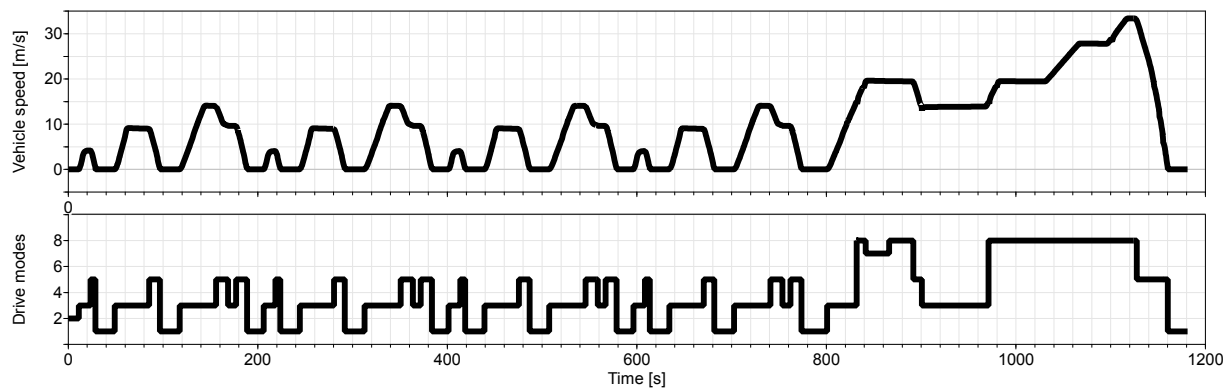


Figure 15: BMW i8 operated as a hybrid vehicle

- [7] C. Schweiger, M. Dempsey, and M. Otter. The PowerTrain Library: New Concepts and New Fields of Application. *The Modelica Association*, pages 457–466, 2005.
- [8] D. Simic and T. Bäuml. Implementation of Hybrid Electric Vehicles using the VehicleInterfaces and the SmartElectricDrives Libraries. *The Modelica Association*, pages 557–562, 2008.
- [9] D. Simic, H. Giuliani, C. Kral, and J. V. Gragger. Simulation of Hybrid Electric Vehicles. *The Modelica Association*, pages 25–31, 2006.

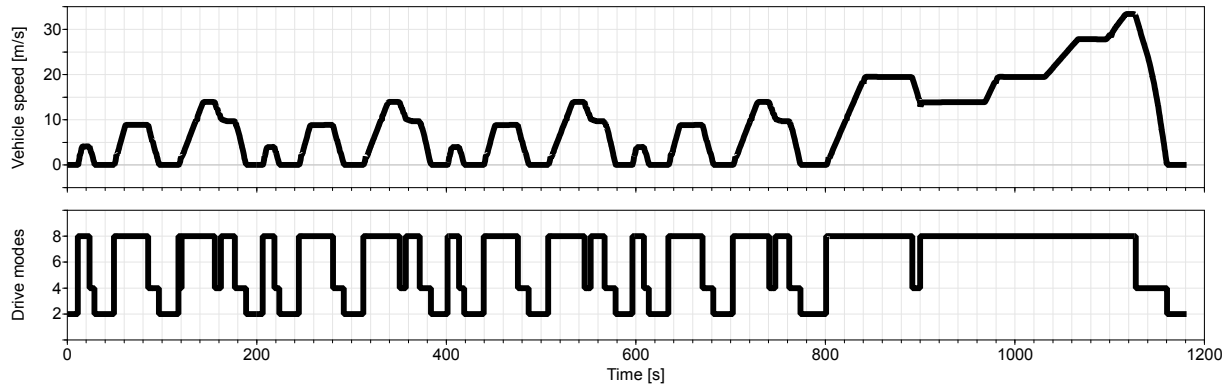


Figure 16: BMW i8 operated as a conventionally driven vehicle

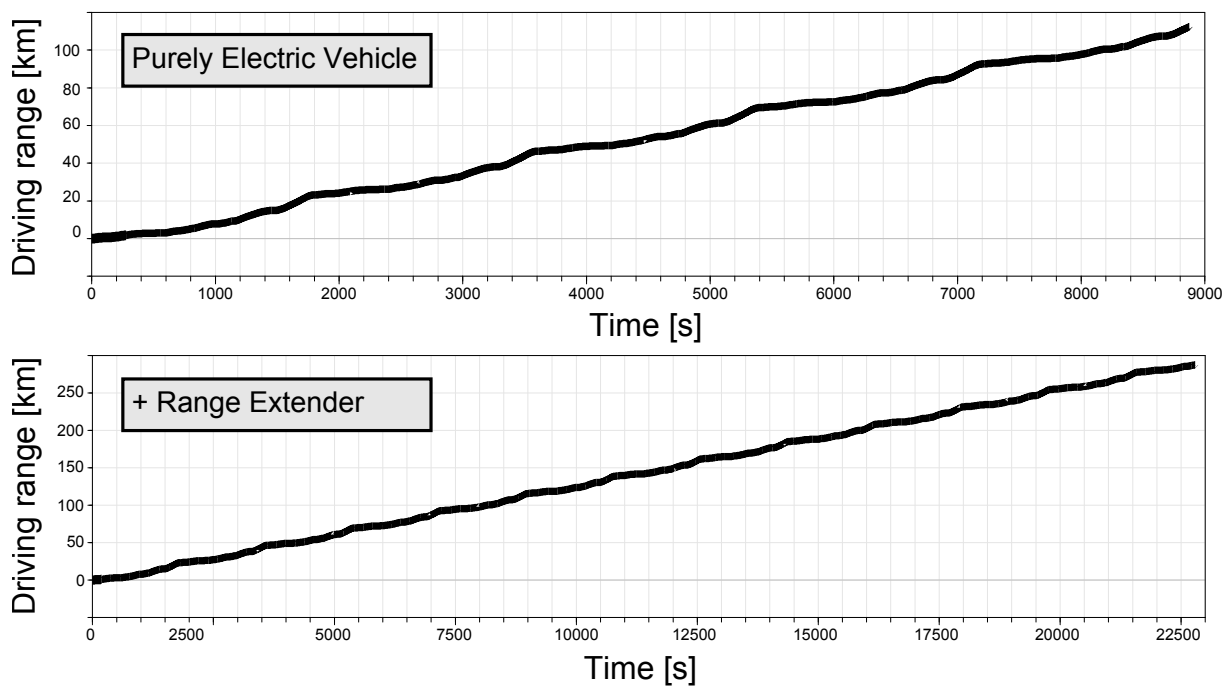


Figure 17: Driving range

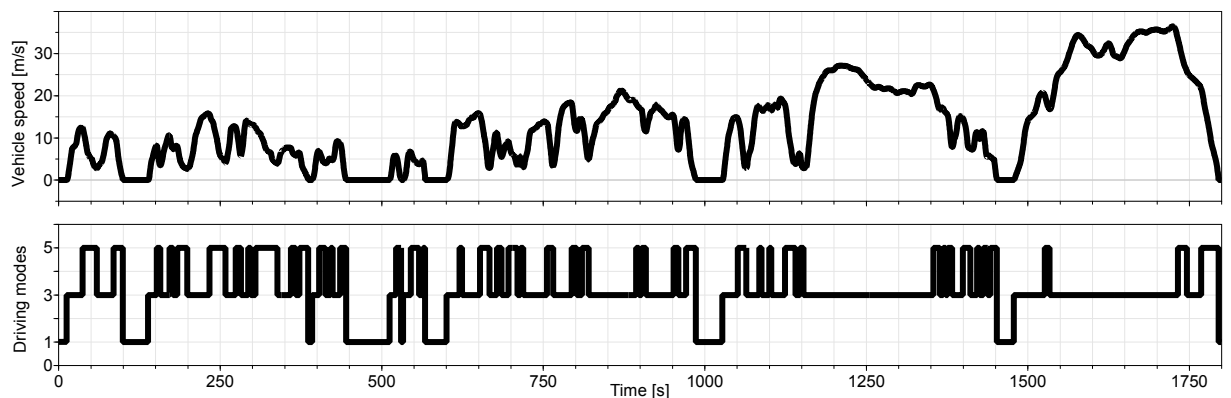


Figure 18: BMW i3 operated as a conventionally driven vehicle



# Thermal shock testing for Engines in Dymola

Alessandro Picarelli<sup>†</sup> Eduardo Galindo\* Gonzalo Diaz\*

<sup>†</sup>Claytex Services Ltd.,  
Edmund House, Rugby Road, Leamington Spa, CV32 6EL, United Kingdom

\*AVL IBERICA SA - OFICINA VALLADOLID  
P. Tecn. Castilla-León, Edif. Centro, E-47151, Boecillo (Valladolid)

<sup>†</sup>alessandro.picarelli@claytex.com

\*Eduardo.Galindo@avl.com

\*Gonzalo.Diaz@avl.com

## Abstract

In this work, we use an acausal multi-domain physical system model to study the interaction between an internal combustion engine operation and a range of cooling scenarios. Although the model can be used for modelling a wide range of scenarios, in this paper we concentrate on the application of “thermal shock”. An internal combustion engine is load-controlled on a dynamometer and coolant temperature transients are imposed on the engine system. Using freely available and commercial Modelica Libraries within the Dymola environment, the whole system integration of the coolant rig and engine dynamometer is achieved. This allows the user to develop and define control strategies for the tests from desktop, prior to engaging in the real tests.

*Keywords: Engine testing, thermal-shock, control system development*

## 1 Introduction

Engines need to work under a variety of temperature conditions. Some engine failure modes are caused by temperature cycling which in turn causes thermal expansion and contraction. This phenomenon can induce mechanical stresses which in extreme cases can lead to component failure.

When an engine starts working it generates heat. This waste heat causes the engine components as well as the oil and coolant temperatures to rise. A thermostatic valve is used to ensure the engine and its fluids reach and maintain the optimum operating temperature for the duration of its use.

The thermostatic valve [1] remains closed during initial engine operation until the coolant temperature exceeds a set point. The thermostat then opens completely and a certain amount of cold coolant flows into the engine. Repeated hot/cold thermal cycles are the cause of the head gaskets failure.

If the cylinder head is kept cooler than the cylinder block, the block and the head will expand at different rates. These non-homogeneous head contractions and block expansions produce strain stresses on the head gasket. Even if the cooling system works well, repeated power on/off cycles will produce cracks on the head gasket surface. The head gasket is the most vulnerable part of any engine and a head gasket failure may result in catastrophic damage to the engine.

The non-homogeneous expansions and contractions of the engine block and head are especially significant with aluminium cylinder heads because aluminium expands approximately two times as much as cast iron when heated. The difference in thermal expansion rates between aluminium head and cast iron block combined with the stress caused by thermal cycles can cause the cylinder head gasket failure.

## 2 Thermal Shock Testing

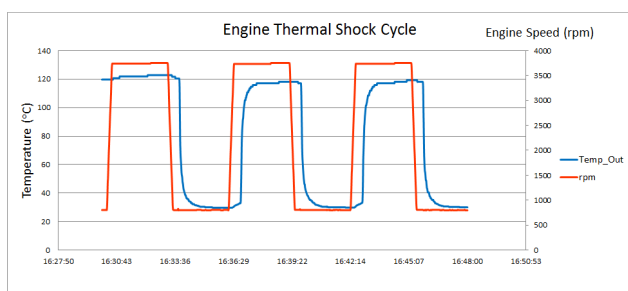
Many manufacturers carry out thermal shock tests to understand and prevent component failure, as well as to accelerate durability testing of engines and engine components, including cylinder-head gaskets.

Thermo-mechanical fatigue is the term used to describe the type of fatigue in which temperature is varied through a cycle. The maximum tensile strain occurs at the same time as the maximum temperature. Maximum compressive strain occurs at the minimum temperature. The main factor causing thermo

mechanical failure is a large number of temperature cycles.

As in fatigue testing, it is possible to accelerate thermal cycling failure modes by increasing the frequency or amplitude of the thermal cycles. These thermal tests are used to simulate critical conditions inside the engine circulating a coolant flow with very large temperature gradients in short periods of time (e.g. 30°C to 120 °C). This is repeated cyclically.

The main task to be performed is to simulate repeated hot/cold thermal cycles. The engine is cycled between rated power and low idle. The coolant is also cycled between hot and cold respectively by means of an external conditioning unit.



**Figure 1:** Example of an engine thermal shock cycle. Engine speed shown in red and coolant outlet temperature from engine shown in blue (°C).

The temperature gradient in the warm up and cooling down cycles is critical to the mechanical stresses applied to the engine due to the thermal shock.

These kinds of tests allow manufacturers to reproduce the whole life of an engine in about 500 hours for a light duty passenger car and 2000 hours for a heavy duty vehicle.

Manufactures expend great efforts in obtaining a good correlation between specific tests and the actual life time of an engine. Once the correlation is completed, the test must be performed as accurately as possible to preserve this correlation.

### 3 Necessity of a Simulation Model

The interaction of an engine cooling system and coolant over time leads to complex equations which need to be integrated versus time in order to predict the cooling and heating ramps the engine will experience.

Adding the thermal shock equipment responsible for the temperature cycle control increases the complexity of the problem.

The requirement to design a system capable of following a specific temperature profile over time with high accuracy, guaranteeing fast temperature gradients, implies the need to develop simulation models capable of dealing with this problem.

Existing simulation tools available within AVL (e.g. AVL boost) do not cater for multi-domain systems engineering where various systems performing different functions are integrated with one another. They are more specialised tools used for simulation particular types of system. When there is the need to simulate such diverse types of systems integration, Dymola through the Modelica language serves its purpose.

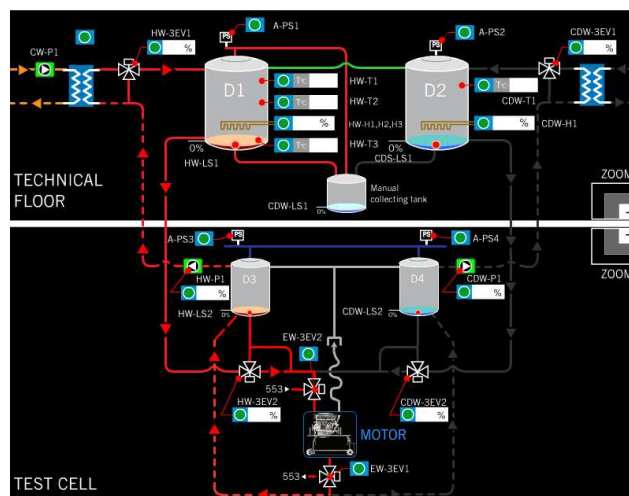
## 4 Case Study

A thermal shock unit is designed to test different Diesel engines with powers rating from 8kW to 150 kW and weights rating from 125kg to 556 kg.

Thermal shock must be performed measuring engine coolant outlet temperature and cycling from 30°C to 120°C.

### 4.1 Thermal Shock Equipment Concept

The equipment consists of two coolant tanks each with a capacity of 1500 litres. One tank is held at a high temperature (120°C) and one at low temperature (30°C). Two additional tanks installed in proximity of the engine and serve as an engine outlet pressure control. Pumps and valves control flow which is circulated and bypassed in the engine creating the thermal shocking of the engine.



**Figure 2:** Example of an engine thermal shock rig showing the main components and routing, including the equipment split between the technical floor and the test cell.

## 4.2 Thermal Shock Testing Equipment

The unit is divided in two main modules, one installed in the technical floor containing the main tanks.



**Figure 3:** Thermal shock rig (technical floor side) with hot (left) and cold (right) tanks.

A second one installed in the test cell with the switching valves and the pumps.



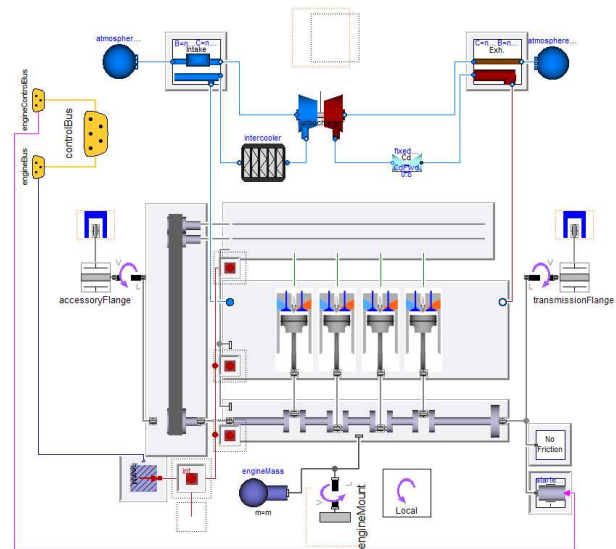
**Figure 4:** Thermal shock rig (test cell side) with test bed (blue), routing valves and pipes on the left.

## 5 Model Development

### 5.1 Engine Model

The engine model is based on the Claytex Engines library which offers a wide range of engine configurations and component detail scenarios [2], [3], [4], [5].

The engine model used in this example is a 2.2l common rail turbo-diesel inline 4 cylinder engine. It is a Mean Value engine model where the intake and exhaust air paths are modelled [6] as well as the emissions, pressure charging, torque generation and fuelling. The engine mechanics is multi-body in type utilising the efficient rotational 3d components from the Claytex library. The Claytex rotational 3d library components allow multi-body modelling with comparable computational efforts to a 1d mechanical system, and have been used throughout our projects.



**Figure 5:** Mean Value engine model with replaceable mechanical and fluidic subsystems.

The engine is coupled to a torque based Engine Control Unit which specifies the fuelling and torque required but also controls ancillaries such as the turbo-charger wastegate.

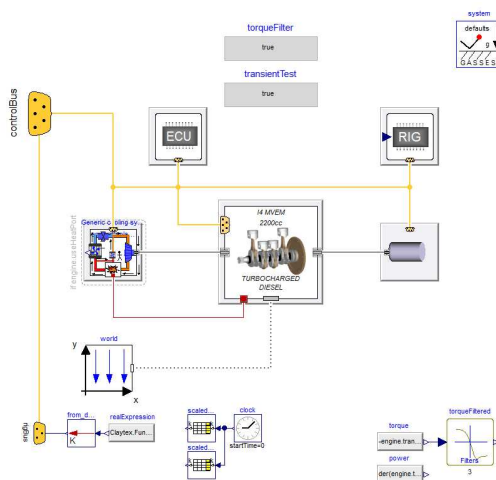
The engine heat release to coolant has been defined as a fraction of the crank power and varies depending on engine speed and load. The fraction value is determined from steady state tests by calculating the power to coolant required for the coolant temperature change between the coolant inlet and outlet of the engine. The coolant path within the engine is represented by a single pipe having average diameter of the passageways and the measured total engine coolant pathway volume and surface area. The pipe diameter is adjusted to achieve the required coolant flow speed through the engine.

The engine thermal mass in this case is a lumped thermal mass and is not split by component. More detailed models are available within the Claytex En-

gines library for studies which require higher level of engine thermal mass discretisation.

The engine dynamometer is controlled via a rig controller which can load or motor the engine according to the experiment requirements.

The turbocharger system is Stodola/Ellipse [7] based and yields a dynamic response. The heat release from the combustion is channelled through 1d thermal ports to the engine coolant system and rig. The coolant pump of the engine is replaced by electric coolant pumps within the rig which can be controlled to deliver specific flows or flow profiles.



**Figure 6:** Mean Value engine model mounted on test bench with coolant system based on thermal shock test rig.

The heat transfer from the engine to the coolant is calculated by means of a Nusselt Number correlation, calculated specifically for this engine. The Nusselt Number correlation is then used within the pipe model which represents the coolant path within the engine. Due to the fact that the thermal mass of the engine is of lumped type, the volume model used to represent the mass of coolant within the engine has one thermal node. The same Nu correlation can be implemented with multiple node fluid pipes derived from the Modelica.Fluids library should a more detailed thermal discretisation be required. The exact same Nusselt Number heat transfer approach is used for the heat exchangers in the rig model described in section 5.2.

## 5.2 Rig Model

The coolant rig must be able to supply preconditioned engine coolant to the engine at two or more different temperatures. The rig described in this pa-

per uses two 1500 litre tanks kept at temperatures with fixed set-points. One tank is kept at a high temperature and the other at a lower temperature. Typically these temperatures are ambient and fully warmed up engine coolant temperatures. The tanks are required to also smooth out and absorb any temperature fluctuations in the rig coolant.

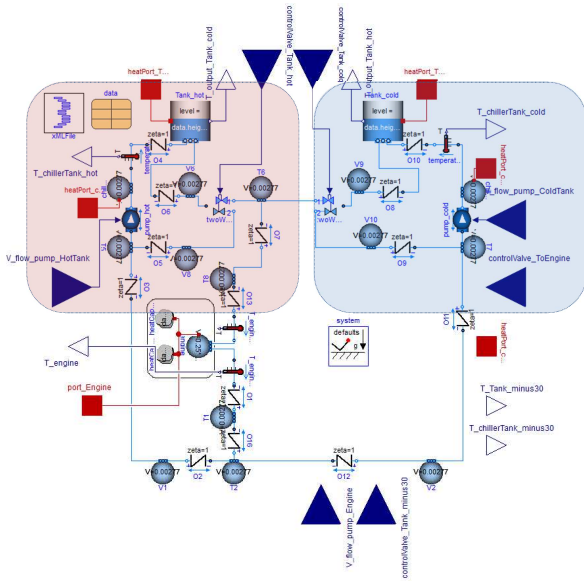
At particular points in the cycle two 3-way valves are controlled to channel either hot or cold coolant through the engine. This change in coolant temperature yields the required thermal shock for the engine to experience and operate through. The tanks should be sized big enough to be able to absorb any fluctuations in coolant temperature even after switch over of the 3 way valves.



**Figure 7:** Example of 3-way valves used in the rig to route the coolant from each of the tanks through the engine.

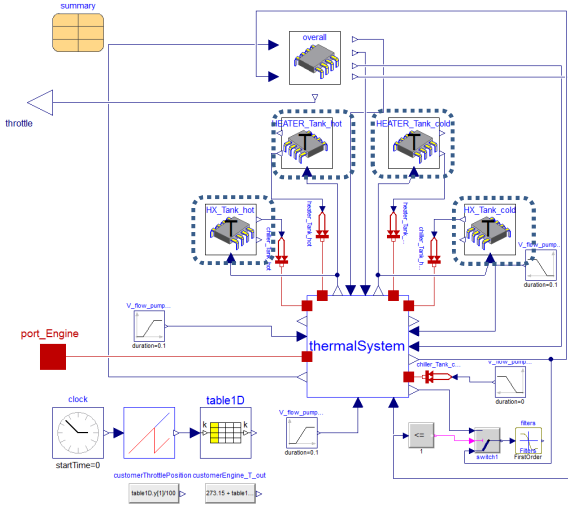
The hot and cold tanks are controlled to their set-points by the use of a water to water heat exchanger (for cooling the fluid down) and a heating element mounted in the tanks for increasing the tank coolant temperature. It is required that the tank temperatures are returned to their desired set-point prior to each tank circuit being re-routed through the engine water jacket.

The rigs are modelled using the Modelica.Fluids and Modelica.Media libraries [7] with some customized components from the Claytex library which incorporates some advanced functionality within the components both for visualization and enhanced model efficiency. The fluid used matches that of the rig in terms of properties and is a mixture of 50% Ethylene Glycol and water with linear compressibility.



**Figure 8:** Thermal shock rig with hot (left) and cold (right) circuits.

The controllers for the tank heaters and coolers are of PID type and tuned to account for the power of these components and the response required of the system.



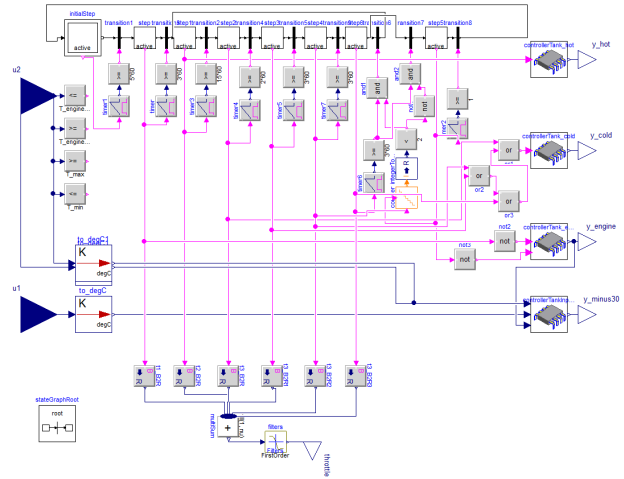
**Figure 9:** PID controllers for controlling the heat exchangers and heaters to maintain tank fluid temperatures close to the set-point.

If the tank fluid temperature has strayed from the set point, the heater and cooler and associated controller need to be sized and parameterised appropriately so that the set-point temperature can be restored prior to the next hot or cold cycle.

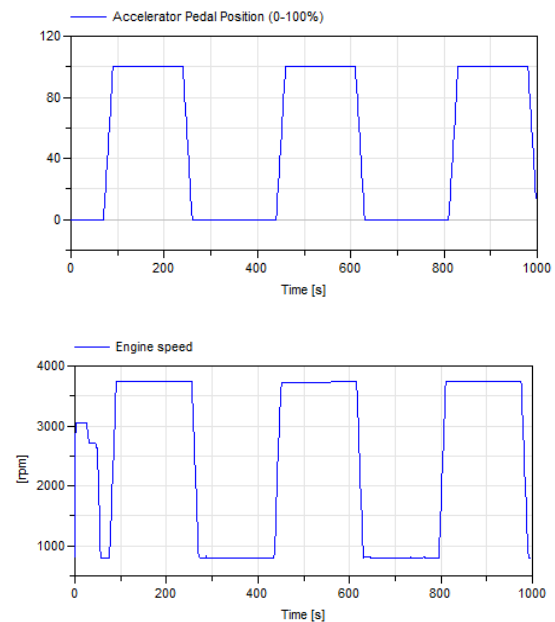
To control the 3-way valves a Modelica.StateGraph model was created (Figure 10). At particular points in the cycle the valves are operated to route the hot or cold coolant through the engine. The same State-

Graph model controls the throttle pedal position which is cycled from 0-100% in a similar phase to the engine speed (Figure 10).

The PIDs for the tank heaters and cooler control were optimised to achieve minimal deviations (1-3 °C) during the cycles.



**Figure 10:** StateGraph controllers for the 3-way valves.

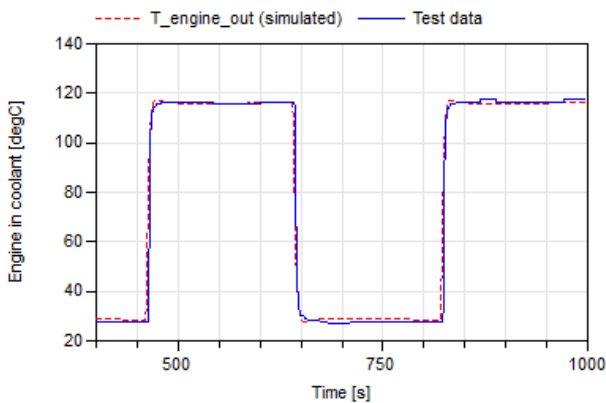


**Figure 11:** Resulting accelerator pedal position (top) and engine speed (bottom) for the thermal shock test.

## 6 Results

The integrated engine and coolant rig, as shown in Figure 5, yielded close results to those measured on the test bed with some minor discrepancies on the

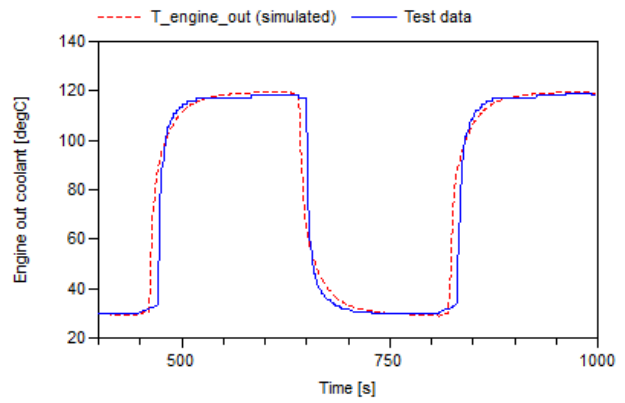
hot part of the cycle when the engine is operating at full power. We analyse the engine coolant in and engine coolant out temperatures as shown in Figure 12. Firstly we consider the accuracy of the inlet coolant temperature to the engine. If the temperature of the coolant at the engine inlet is not accurate then this will influence the exit temperature and potentially reduce the accuracy of the model. Discrepancy in the coolant inlet temperature after ramp up and ramp down ranges from 0.4-0.85°C. This type of error is deemed acceptable and demonstrates that the conditioning system is sized and modelled correctly.



**Figure 12:** Engine coolant inlet temperatures for test data (blue) and simulated data (red dashed).

Being satisfied with the coolant inlet results we now concentrate on the engine coolant outlet results. This is the coolant temperature exiting the engine. We notice the ramp downs (shown in Figure 13) at  $t \sim 250s$  and  $t \sim 650s$  are offset in time. Initially we get a faster cooling of the modelled engine but as the coolant temperature approaches the cold circuit temperature (27 °C) the modelled coolant temperature transient slows down. Nevertheless we note that the lower temperature is achieved prior to the coolant temperature increase ramps at  $t \sim 470s$  and  $t \sim 830s$ . On the temperature increase ramps the engine coolant heats up marginally quicker than the test shows and overshoots the test data at the higher coolant temperatures  $t \sim 600s$ ,  $t \sim 950s$  by approximately 0.3-1°C.

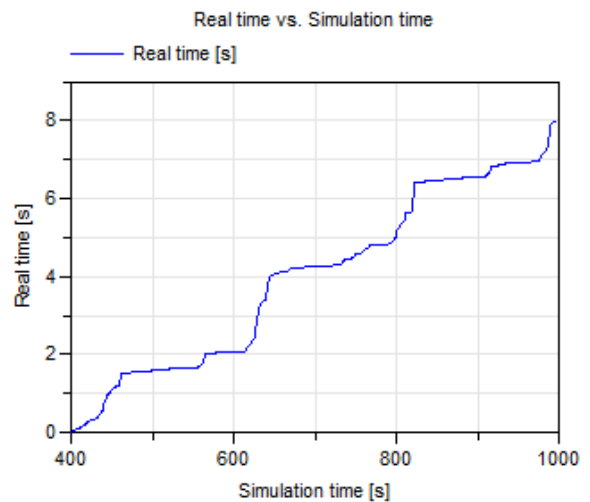
The high temperature sections correspond with the engine being at full load and speed. The overshoot tells us that there is excess heat being put into the coolant by the engine at maximum power in the model.



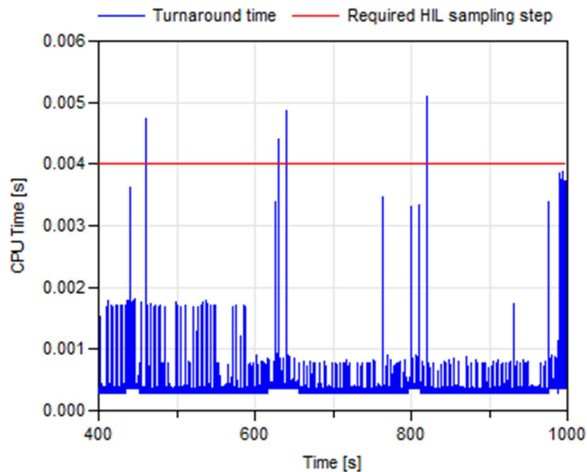
**Figure 13:** Engine coolant outlet temperatures for test data (blue) and simulated data (red dashed).

The minor discrepancy in the transients is believed to be linked to the correction required for heat rejection of the engine to the coolant and potentially the thermal mass of the engine itself.

The whole system model on average is shown to run faster than real-time with both variable (Figure 14) and fixed step (Figure 15) solvers, the latter making it suitable for hardware-in-the-loop simulation. For fixed step solver the target turnaround time is 1ms. There are occasional spikes denoting overruns (Figure 15) which can be accommodated for. Further improvements in event checking suppression should allow the reduction if not removal of these spikes.



**Figure 14:** Model performance using Radau IIA variable stiff solver for stiff systems showing Real time (vertical axis) vs. simulation time (horizontal axis).



**Figure 15:** Model performance using Euler fixed step solver and implicit inline integration showing CPU time per calculation step (vertical axis) vs. simulation time (horizontal axis)

## 7 Conclusions

This paper shows how a highly complex multi-domain system to reproduce engine thermal cycling scenarios such as “Thermal shock” can be created within Dymola. Detailed physical representation of the internal combustion engine as well as the coolant rig, valves and controllers has been achieved. The acausal and multi-domain properties of the Modelica language make it possible to simulate all these integrated systems within one larger system model without the need to interface with third-party tools.

Despite the model complexity it will run faster than real-time. Symbolic manipulation plays a large factor in this reducing the number of equations to be solved from in excess of 27000 to just over 3700.

The model will allow the user to specify and define appropriate components and control strategies used to test engines without actually having to use a real engine. This in turn reduces costs, pollutant emissions and the time involved in calibrating a rig and controller with the internal combustion engine fitted and run.

Lastly, the model has been shown to achieve real-time simulation with Euler fixed step solver and implicit inline integration making it suitable for HIL applications.

## References

- [1] Mitchell T., Salah M., Wagner J., Dawson D. (2009) Automotive Thermostat Valve Configurations: Enhanced Warm-Up Performance ASME Journal of Dynamic Systems, Measurement and Control.
- [2] Dempsey M., and Picarelli A. (2009). Investigating the multibody dynamics of the complete powertrain system. Como, Italy: Proceedings 7th Modelica Conference.
- [3] Dempsey M., Picarelli A, Fish G. (2012). Using Modelica models for driver-in-the-loop simulators. Munich, Germany: Proceedings 9th Modelica Conference.
- [4] Dempsey M., Roberts N. (2012) Predicting the launch feel of automatic and dual clutch transmissions. Munich, Germany: Proceedings 9th Modelica Conference. Yang Y., Lam R., and Fujii T., (1998) Prediction of torque response during the engagement of wet friction clutch. SAE Technical Paper, 981097.
- [5] Dempsey M., Roberts N., Picarelli A. (2013) Detailed Powertrain Dynamics Modelling in Dymola – Modelica. IFAC-AAC conference Tokyo, Japan.
- [6] Hendricks et al. (1996) Modelling of the Intake Manifold Filling Dynamics SAE 960037.
- [7] Stodola, A. (1945). Steam and Gas Turbines. McGraw-Hill, New York. Reprinted by Peter Smith.
- [8] Casella. F. et al. (2006) The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks Modelica Conference, 2006





# Model-Based Design of Integrative Energy Concepts for Building Quarters using Modelica

Dipl.-Ing. Torsten Schwan                      Dipl.-Ing. René Unger  
EA Systems Dresden GmbH, Königstr. 2, 01097 Dresden, Germany  
torsten.schwan@ea-energie.de    rene.unger@ea-energie.de

Dr.-Ing. Christian Lerche  
Ingenieurbüro Dr. Lerche, Lugstr. 5, 01796 Pirna, Germany  
doclerche@web.de

Dipl.-Ing. Christian Kehrer  
ITI GmbH, Schweriner Str. 1, 01067 Dresden, Germany  
kehrer@itisim.com

## Abstract

Increasing energy prices as well as outdated building systems present the housing industry with the challenge of finding new complex system solutions including renewable energy and storage systems. The municipality Lohmen and the local housing association contracted EA Systems and IB Dr. Lerche to develop an integrative energy system concept for its historic town center.

This paper deals with modeling and simulating different energy system variants for the existing building structure using the Modelica-based ‘Green Building’ library and SimulationX. The discussion illustrates the challenges of the modeling process, innovative solutions and the simulation results.

*Keywords: Green Building, Building Simulation, Building Quarters, Building Complexes*

## 1 Introduction

In Germany, most buildings were erected before any heat insulation regulations existed. Therefore, most of them had to be refurbished from time to time to reduce thermal energy consumption and to adapt to current legislative requirements (e.g. Energy Saving Ordinance).

Especially in the East of Germany, last region-wide refurbishment measures were taken back in the 90s of the last century. Hence, most buildings there still meet current energetic requirements. However, the building energy systems installed at that time have a lifespan of about 20 years. Consequently, building owners today, whether they are private or public, have to think about new refurbishment measures.

Conventional measures to replace existing, outdated heating systems with more efficient system components (e.g. gas-fired condensing boilers) contribute to a significant reduction of energy consumption. But rising energy prices and upcoming legislative requirements to reduce carbon dioxide emissions mean that these measures will not be sufficient. Innovative ideas, like integrating renewable energy systems and storages, have to be tested for applicability to specific building situations.

To meet this challenge, the municipality of Lohmen and the local housing association contracted EA Systems and IB Dr. Lerche to develop an integrative energy system concept for the historic town center. This requires local characteristics, especially the availability of renewable energy, to be scrutinized. In this regard, suitable energy system variants have to be derived and evaluated for energy efficiency, economical requirements and further ecological aspects.

The given complexity requires new analytical methods. The availability of renewable energy has to

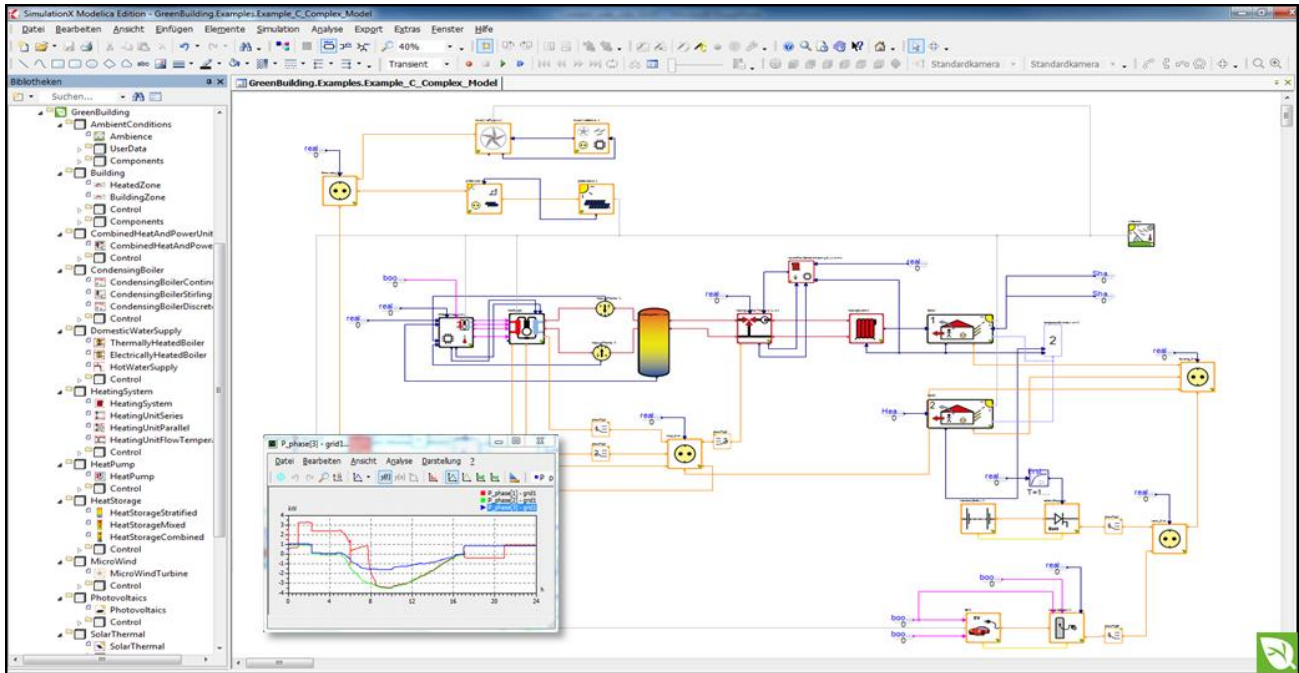


Fig. 1: Modelica model of a renewable energy system incl. the building in the simulation environment

be analyzed in combination with strongly usage-dependent energy consumption profiles as well as state-specific applicability of partly new storage systems (e.g. batteries). For that reason, only dynamic simulations of different energy systems can provide sufficient results for an adequate system evaluation.

As a newly developed simulation environment, the Modelica-based ‘Green Building’ library and SimulationX were used to fulfill the given engineering task. Since it was developed especially for such applications, ‘Green Building’ can be used to easily model complex energy systems including renewables, storages as well as complex building structures in one simulation environment.

## 2 Green Building Simulation Library

Modelica is an equation-based and cross-domain modeling language. It offers the possibility to model complex building energy systems with different domains (e.g. heat, electricity, control) using differential-algebraic equations. These equation systems can be edited and solved within one simulation environment.

That is why EA Systems used Modelica and the versatile CAE tool SimulationX to develop the ‘Green Building’ library for the simulation of energy systems [4]. By adapting an approach widely used in the automotive industry, several elements for the production of renewable energy and heating systems

were created as well as storages and electrical or thermal consumers [5]. Most of the models represent real world objects like vehicles, electrical inverters or valves. Granularity and complexity of each element are thus in the same range while preserving a flexible yet easy modeling process (i.e. physical as well as phenomenological models [5]). The modeling focus lies on the interactive behavior of different energy system components with varying complexity in the context of building energy supply, be it thermal or electrical (i.e. electrical systems modeled using RMS values [4]). Although the building itself can be modeled as a complex thermal and electrical energy consumer by using a number of thermal zones, a detailed thermal building simulation for different thermal conditions in one room, for example, requires a more specialized tool, like EnergyPlus (Green Building, for example, uses constant average temperatures in thermal zones [5]).

Figure 1 shows the ‘Green Building’ simulation environment in ITI’s SimulationX with a relatively simple model of a single-family home including a heat pump as well as a micro-wind turbine and a photovoltaic system as renewable energy sources [6].

## 3 Municipal area of Lohmen

The surveyed area in the municipality of Lohmen, a small town in the middle of the Free State of Saxony in Germany, includes altogether twelve buildings

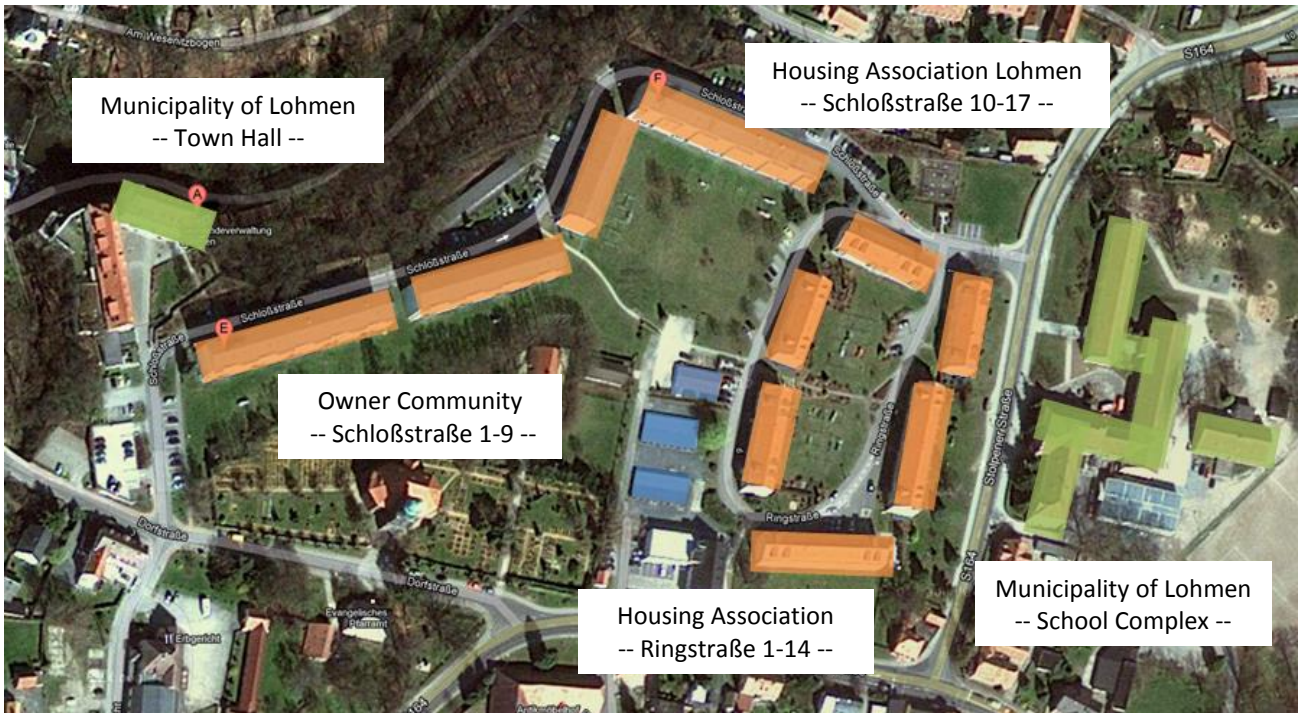


Fig. 2: Historic town center of Lohmen

and building complexes. Besides ten dwelling houses built in the 1960s and 70s, there are also the over 500-year old Lohmen castle, the office of municipality, as well as the school complex with four separate buildings which are part of the study.

Figure 2 shows the historic town center of Lohmen with all acting partners, its municipality, the housing association as well as a local owner community.

Intensive analyzes of available energy saving potentials and existing renewable sources were the basis of the following modeling and simulation work. Initial analyzes dealt with conventional refurbishment measures of existing heating systems and the usage of solar energy. Besides, Lohmen is directly situated at the river Wesenitz with a number of existing water power plants in the area of the historic town center. Hence, further analyzes mainly revolved around the integration of water power as well as water heat usage with heat pumps, for example.

#### 4 Modeling paradigms and simulation results

Basically, energy can be provided in two ways. Heat and electricity can be supplied to each house individually. That means heat is produced by internal heating systems, like condensing boilers or micro-generation units. Electricity can be taken from the grid or from locally installed photovoltaic systems

on each building's rooftop. These basic approaches are single-building solutions which only replace or modify existing building energy supply concepts. That way, existing renewable energy potentials (e.g. river water heat) cannot be used in an economic way.

Other solutions consider all existing buildings together. These quarters solutions enable an economical use of all available renewable energy potentials as well as existing synergy effects, e.g. electricity production with photovoltaic systems only on rooftops facing South.

Basically, SimulationX' 'Green Building' library was used to implement and to simulate adequate system models. This simulation environment enables the user to simulate variable building energy systems including a building's heating as well as the inhabitants' electricity consumption. The basic modeling idea though was a single-family home. That required some modifications of existing models and the development of new, adequate modeling approaches for the project at hand.

Figure 3 shows two of these newly developed modeling paradigms. The model on the left shows the school complex which consists of four building models for each part of the complex (elementary school, kindergarten, gym, historic school). Each of these building models is implemented as three 'Building Zones' from the 'Green Building' library representing basement, classrooms and roof. Furthermore, the school complex model includes domestic water production, electricity consumption as

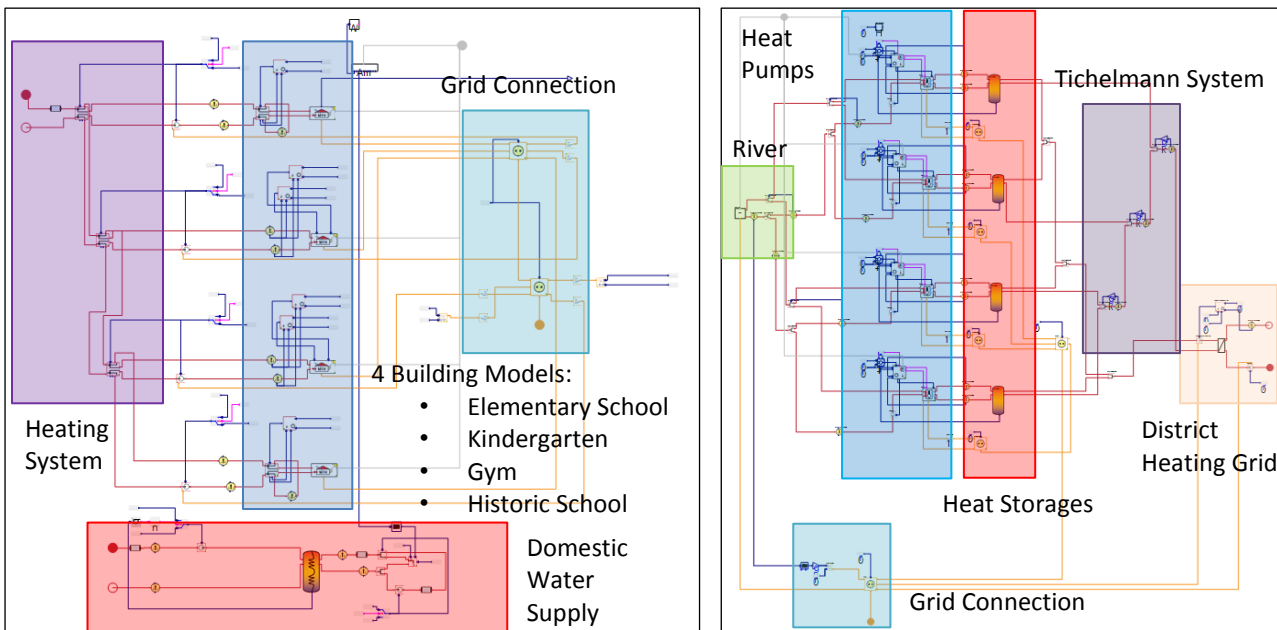


Fig. 3: Sub-model structure of the school complex and the heat pump cascade

well as heating system components (e.g. valves, pipes, etc.).

The right side of figure 3 shows the implemented sub-model of a heat pump cascade to be used as a basic heating system for the quarters. It consists of ‘Green Building’ heat pump models including connected controllers as well as heat storages. Furthermore, a river model provides information about available water heat and renewable electrical power of existing water power plants. Heat supply to the district heating grid is represented by a convenient heat exchanger model.

Both modeling approaches illustrate the basic modeling idea used during this project. Buildings and usage (domestic water consumption, electricity consumption, etc.) are implemented individually in sub-model components. That way, they can either be used for single-building simulations or for entire quarters simulations without the need for further model adaptations.

#### 4.1 Single-building simulation

The following energy system configurations including renewable energy usage were examined in single-building analyzes:

- Condensing boilers
- Micro-cogeneration
- Photovoltaic systems
- Solar thermal collectors
- Gas-fired heat pumps with geothermal energy usage

The replacement of existing condensing boilers always represents a basic strategy to lower natural gas consumption of each building. They are able to provide heat for buildings and domestic water supply at fairly high efficiency rates (higher than 95%). Adding small micro-cogeneration units to each building’s energy system allows for a combined heat and electricity production. That way, general electricity consumption, e.g. for heating pumps and hallway lighting, can be significantly reduced. Especially in Germany, cogeneration units are substantially subsidized by the state. Hence, heat and electricity can be produced at relatively low operating expenses.

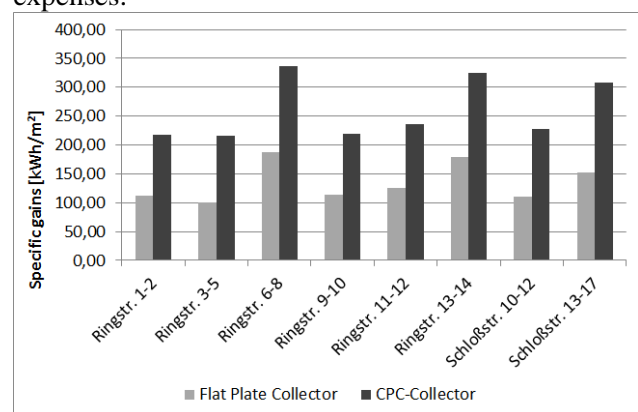


Fig. 4: Specific renewable heat production of different solar collectors at different residential buildings

Photovoltaic systems provide renewable electricity to meet a building’s general electricity demand. In Lohmen, general electricity consumption is rather low. Therefore, local electricity production has to

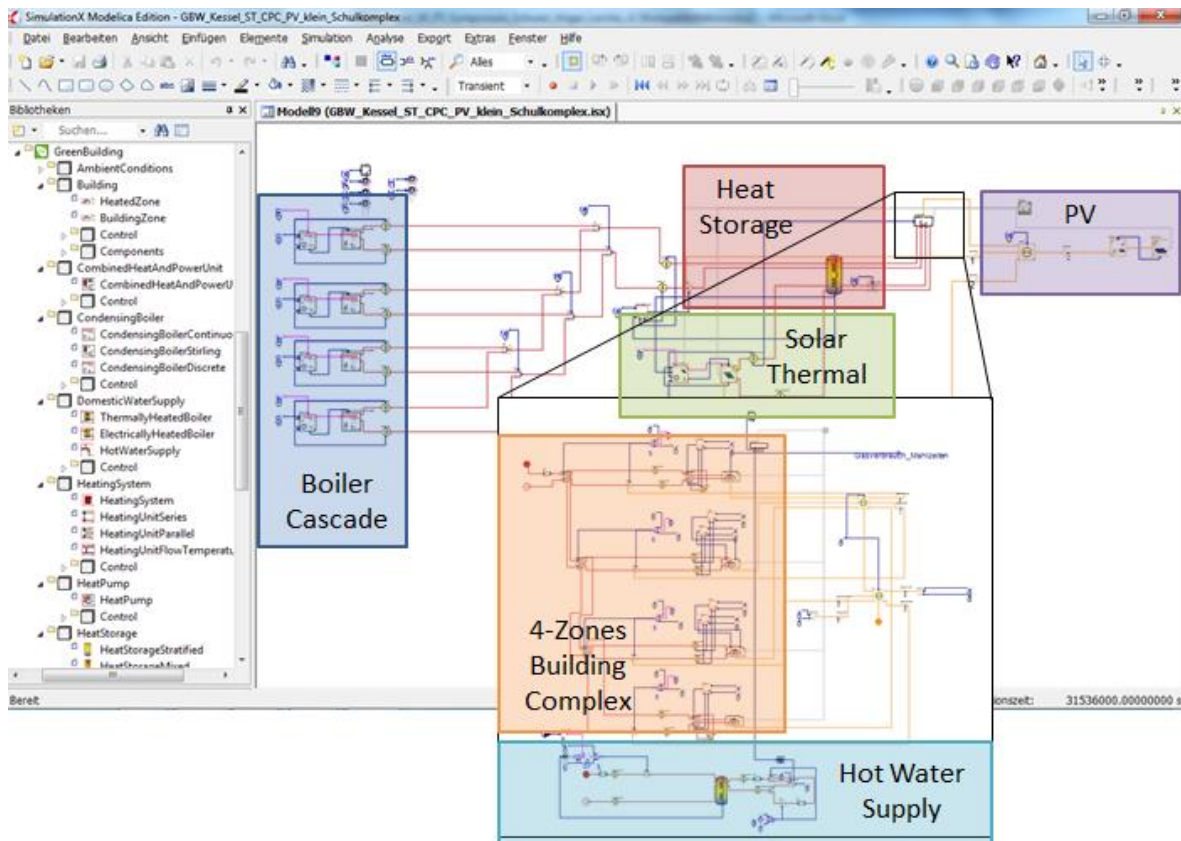


Fig. 5: Single-building model of the school complex with condensing boiler cascade, heat storage, solar thermal collector and photovoltaic system

account for these conditions. The inhabitants' electricity consumption is much higher, but cannot be met easily by local electricity production, because of German tax legislation for housing associations. Furthermore, electricity fed into the grid is not very profitable for newly constructed energy systems. Consequently, the main goal of energy system design is to minimize the amount of electricity fed into the grid while maximizing the production of renewable energy.

Solar thermal collectors constitute another energy system configuration that uses solar energy as a source. They can provide a significant amount of a building's domestic water consumption if they are installed in the right spot (e.g. south facing roofs). However, there are two different types of collectors: flat plate and CPC (Compound Parabolic Concentrator) collectors. Figure 4 shows some sample results from simulation-based analyzes of the solar thermal energy potential of various residential buildings' roofs. CPC collectors always represent a more efficient system variant. Due to quite high system costs, a specific heat gain of at least  $250 \text{ kWh/m}^2$  is needed on average. Figure 4 shows that only CPC collectors on buildings with south facing roofs (e.g. Schloßstr. 13-17) can provide enough solar energy input for the

analyzed system configuration (domestic water production with hot water boilers at  $60^\circ\text{C}$ ).

The third energy system configuration that was simulated and evaluated for each single building is based on gas-fired heat pumps. Compared to electric heat pumps, these comparatively new systems provide heat using geothermal energy as well as a smaller amount of natural gas.

Figure 5 shows a complete single-building model of the school complex with a condensing boiler cascade as a basic heat supply system, solar thermal collectors to support domestic water supply, and a photovoltaic system to provide renewable electricity. This sample model shows only one of nine simulated and examined system configurations for the school complex:

- Existing building with two 15-year old condensing boilers
- Refurbished energy system (e.g. increased pump efficiency) with some adaptations of the building insulation
- New condensing boiler cascade with modulating working burners
- Condensing boiler cascade with small Stirling CHP
- Condensing boiler cascade with CHP

- Condensing boiler cascade with solar thermal system (CPC)
- Condensing boiler cascade with CPC collectors and small photovoltaic system
- Condensing boiler cascade with small photovoltaic system
- Condensing boiler cascade with large photovoltaic system

Figure 6 shows results for simulated electricity and natural gas consumptions of the school complex' analyzed system configurations.

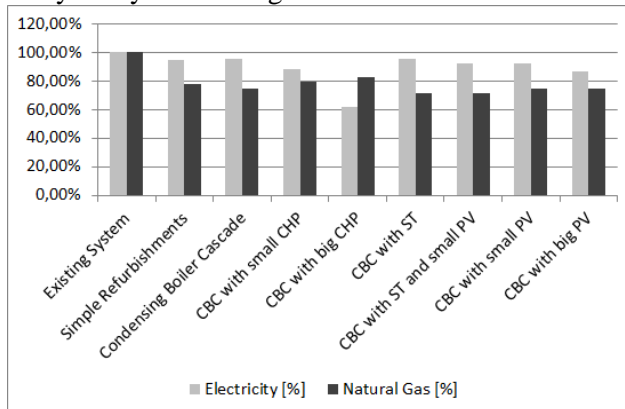


Fig. 6: Simulation results for different system variants of the school complex

All results are compared against the results of an existing old condensing boiler cascade. All variants, except for the existing system, are simulated with simple refurbishments of the building's heat transfer system (e.g. pumps) and the building's insulation. It is evident that even simple refurbishment measures have a significant impact on the heat and electricity consumptions. However, variant 3 with the new condensing boiler cascade is the first energy system variant suitable for long-term decisions due to the age of the existing boiler.

Furthermore, it appears that system variant 5 with a large combined heat and power unit (CHP) could be the most interesting variant regarding energy consumption and system efficiency, because the electricity consumption can be reduced significantly without increasing the natural gas consumption too much.

All in all, over 70 variants of different energy system configurations and different buildings were modeled and simulated using the 'Green Building' library and further presented modeling paradigms. Such energy supply variants for single-buildings, however, have only limited potential to reduce the renewable energy usage and energy consumption. Further analyses have thus been conducted regarding a number of different energy system variants for the entire quarters.

## 4.2 Simulation of the Quarters

The basic idea is to combine all implemented building models (see right part of fig. 3) of ten residential buildings, Lohmen castle and the school complex into one large simulation model plus various energy system configurations. The main part of the heat supply regards a district heating grid implemented with 'Green Building' components. This district heating grid includes volume flows as well as temperature behavior in single pipes to enable dynamic heat loss simulations (i.e. simulated heat losses to ground using a static ground temperature model [7]). The layout of the circulation pump, however, (regarding pressure drops, for example) is configured during preprocessing in order to reduce the number of simulated system states.

Figure 7 shows a sample district heating grid model with several centrally located condensing boiler cascades, a heat pump cascade using river water and water power plants as heat and electricity source as well as two cogeneration units for overall domestic water supply located in the school complex.

Two different types of district heating grids were implemented and analyzed. A "small" district heating grid provides heat from two cogeneration units to meet domestic water requirements of each building. Mechanical space heating is still provided locally by several condensing boiler cascades.

The "large" district heating grid solution includes cogeneration for domestic water supply as well as main heat power production by a 520 kW heat pump cascade. Heat pumps use river water as environmental heat source and electricity produced at local water power plants as electricity source. Both heat sources are connected to each building via a widespread district heating grid. Further heating power demands are met by smaller peak-power condensing boiler cascades.

Modifications can be done by adding further, smaller photovoltaic systems. Additionally, some of the implemented condensing boilers can be replaced by gas-fired heat pump systems again. That way, the system can be optimized with respect to number and type of the required components.

Such a complex energy system requires sufficient control algorithms to operate the appropriate system components. That way, standby losses of system components that are not in use (e.g. peak power condensing boilers at higher ambient temperatures) can be significantly reduced. Furthermore, renewable energy usage can be increased by avoiding simultaneous heat pump operation, for example.

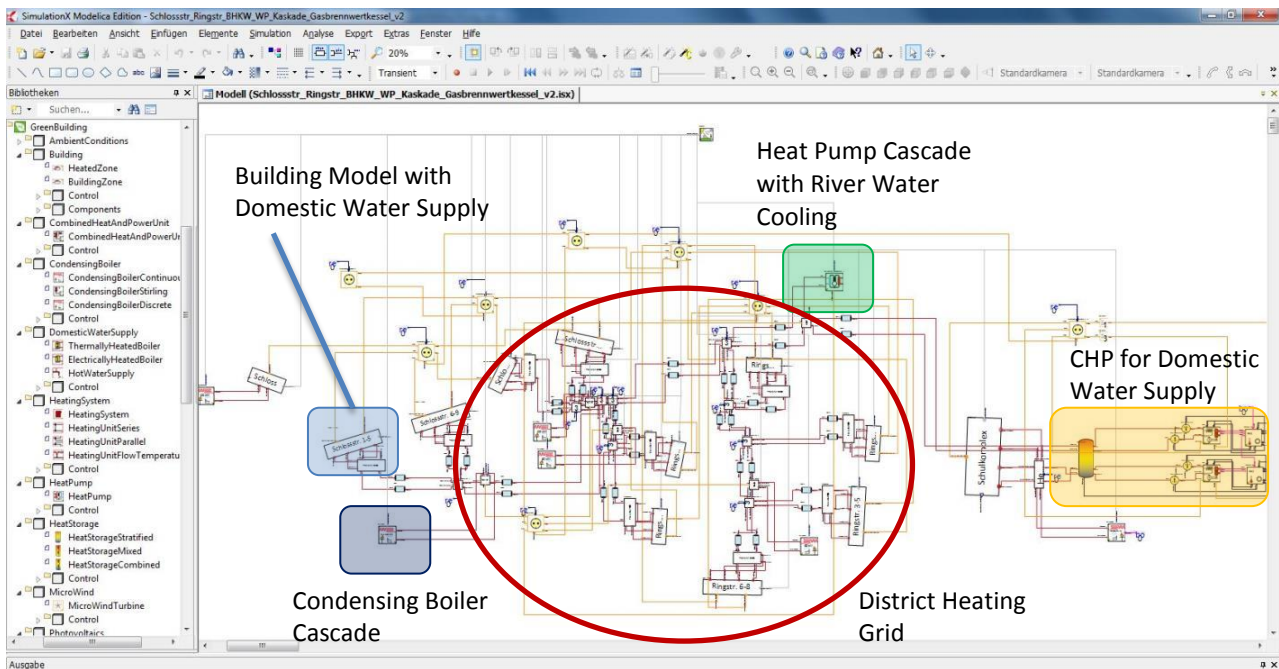


Fig.7: Simulation model of the quarters with district heating grid, micro-cogeneration, heat pump cascade, peak power condensing boiler cascades and all involved building complexes

‘Green Building’ enables the user to implement physical energy systems as well as corresponding controller behavior (mostly P controllers with hysteresis switch-on/off statements) in one large simulation model within the SimulationX modeling environment. Previously presented optimization tasks can thus be executed more easily.

Each of the two types of energy system configurations for the district heating grids were simulated with four and five different system component configurations and/or control algorithm sets. All results were evaluated in comparison with the buildings’ existing energy supply infrastructure and the accumulated results of single building simulations with condensing boilers as the basic heat supply variant.

The simulation’s result analysis focused on the affordability of different energy system components. Cost analyses, however, are not a basic part of the ‘Green Building’ simulation environment. Further result evaluations revealed some interesting aspects regarding energy saving potential as well as overall system sustainability.

Figure 8 shows simulation results for the relative overall energy consumption of each simulated system configuration, divided into natural gas and electricity consumption.

The diagram is divided into three basic parts. The left side shows accumulated results of all analyzed single-building simulations. It illustrates the overall energy consumption of the quarters’ existing energy systems, simply refurbished energy systems as well as the basic “energy-concept” regarding the replace-

ment of old boiler systems by new condensing boilers.

The second part of the diagram shows results for the previously introduced “small” district heating grid variant. Besides basic system configurations, it was also simulated with a 14.5 kWp photovoltaic system on a suitable building roof. Furthermore, the integration of street lighting into the overall energy system concept was part of some analyses, and some of the integrated condensing boilers were replaced by new gas-fired heat pump systems.

Most simulations for the quarters’ analyses were run for a second district heating grid system variant. This “large” variant mentioned earlier includes a heat pump cascade for main heating power supply. It was simulated with and without recognizing street lighting. However, most of the simulated results concern different optimization levels of energy management system.

On the one hand, basic energy management system algorithms forced the CHP cascade to provide heat for domestic water supply in residential buildings even in winter months when sufficient heat power is directly needed in the school complex. On the other hand, all peak power condensing boilers stay activated even in case of ambient temperature levels above the estimated bivalence point. Furthermore, all heating systems, even the heat pump cascade, are operated at comparatively high flow temperature levels.

Evidently, the system does not work in an optimal way. Therefore, some optimization steps were taken

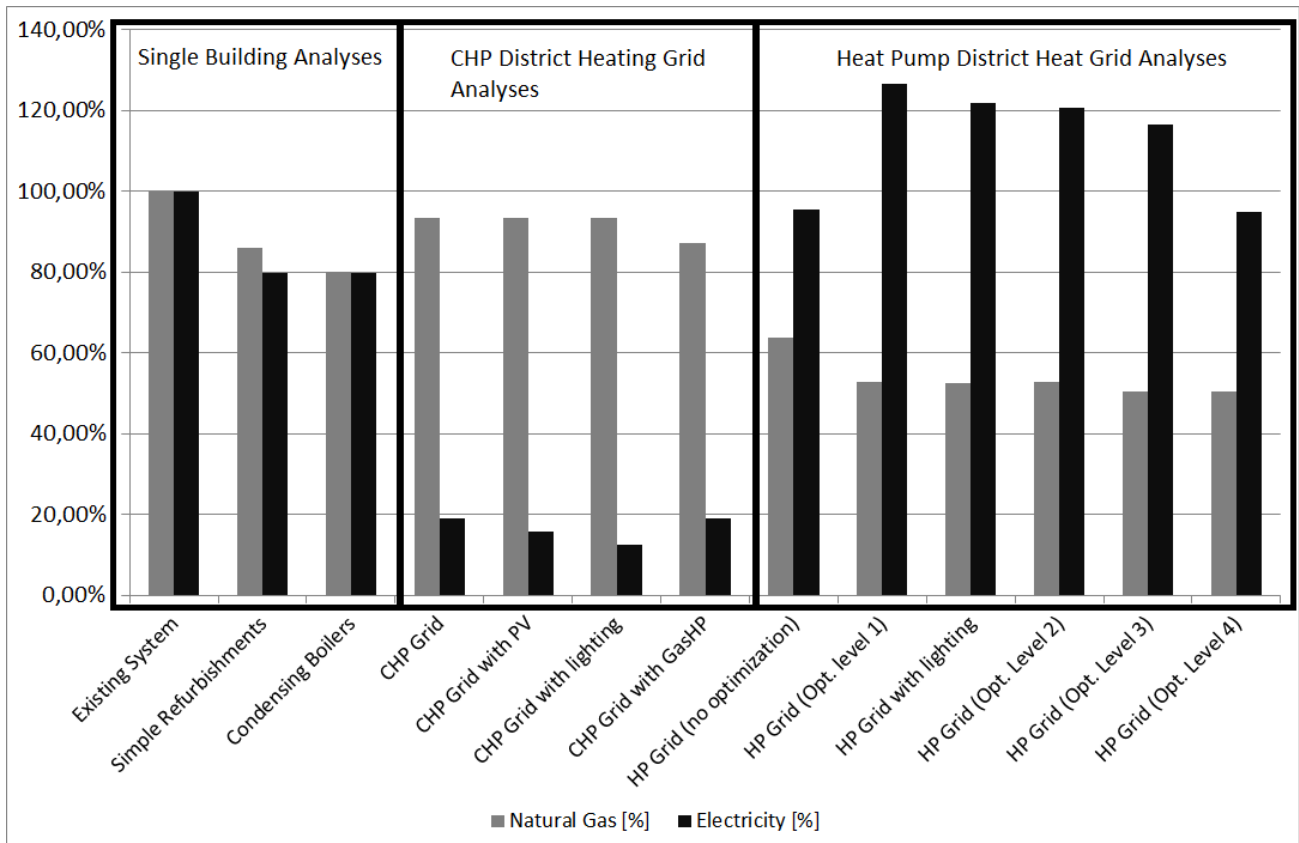


Fig.8: Simulation results of different energy system variants for complete quarters solutions

to improve the implemented energy management algorithms. First of all, the condensing boiler cascades were shut down in case of ambient temperature levels rise above the bivalence point. Furthermore, the CHPs are only used to provide residential buildings with domestic water in transitory and summer periods. These measures are shown as optimization level 1 in figure 8.

Optimization level 2 refers to the reduced operation of the heat pump system during the summer months. Heat pump systems are supposed to provide heat only to connected buildings. However, they are not intended to provide domestic water because of required higher temperature levels. Lowering the heat pump cascade thus reduces standby losses without affecting any heat power needs.

The third optimization level refers to reductions of preconfigured reference temperatures for all implemented heating system components (i.e. heat pump cascades, peak-power condensing boilers). That way, reference temperatures could be reduced by about 5K to enable all components to run in better working ranges (e.g. COP of heat pumps).

The last optimization level affected only switch-on and switch-off times of the planned heat pump cascade. In the planned “large” district heating grid system variant, all heat pumps are equally used to

provide heat (c.f. Tichelmann system in fig. 3). All previously presented energy management algorithms forced each heat pump to work mainly at the same time (i.e. control hysteresis was implemented in the same way). Hence, this control regime was adapted to sequentially drive the heat pump cascade. This way, electricity provided by the water power plant can be used more efficiently to power each heat pump. That enabled a reduction of the overall electricity consumption.

Both variants of district heating grids offer massive reductions of energy consumption and a better use of available renewable energy. The “small” district heating grid variant provides huge reductions of electricity (ca. 80%) consumption even in comparison to refurbished single-building system variants. This is mainly caused by comparatively huge amounts of electricity produced by the CHP system. However, the overall natural gas consumption increases slightly because of partly simultaneous heat and electricity production.

The “large” district heating grid shows opposite results. Using environmental energy as well as electricity to provide heat massively decreases the natural gas consumption (ca. 50%). However, existing power plants cannot generate all electricity required for the heat pumps’ operation. Hence, the overall





Fig.9: Pictures of residential buildings (top left), historic school (bottom left) and ancient Lohmen castle (right)

electricity consumption can even exceed the existing electricity demand. Apparently, system optimization as well as iterative improvements of implemented energy management algorithms can sufficiently decrease these effects.

Finally, both district heating grid solutions provide extensive reductions of energy consumption. This is mainly achieved by using huge amounts of renewable, environmental energy (e.g. water power) as well as synergies in the existing quarters. That way, providing electricity, which is produced by a newly installed CHP system, to street lighting would further decrease overall electricity consumption in a significant way.

Furthermore, replacing some of the additionally needed peak-power condensing boilers by an adequate number of gas-fired heat pumps using geothermal energy as additional heat source can significantly improve natural gas consumption as well.

## 5 Conclusions

This paper shows how the SimulationX ‘Green Building’ library was used to analyze and to evaluate different suitable building energy system variants of building quarters in the historic town center of Lohmen. Some selected modeling paradigms are presented to demonstrate how to use ‘Green Building’ for simulating larger building complexes.

Furthermore, the developed analyses approach, be it the comparison of different energy system configura-

tions for single buildings or the evaluation of the given quarters as one unit, is presented as well.

Some simulation results are illustrated for one single building evaluation process regarding the school complex. In comparison to that, both developed energy system variants representing a district heating grid are introduced with their specific characteristics. Simulation results are evaluated in comparison to single-building analyses. Advantages and disadvantages of both variants and derived system configurations are discussed at the end.

The simulation models are relatively complex. The simulation of a single building energy system took 0.5 to 4 hours for one year. In comparison to that, simulations of the quarters took between 3 and 4 days to finish. It is thus evident that ‘Green Building’ can be used to economically analyze complex variants of building energy systems. Parallelization by using multi-core computing can further improve this aspect.

Extensive engineering efforts yielded a comparatively high numerical stability of the Green Building library [6]. That way, simulating even complex, coupled numerical systems (linear and nonlinear) does not affect convergence. The presented project, for instance, comprised complex quarter models with up to 3000 system states. Simulating these models for one year causes around one million event iterations.

The presented results are part of a research study which is funded by KFW (Development Loan Corporation) and communal as well as private investors. The complete research project includes theoretical,

simulation-based analyses of sufficient energy system configurations. First approaches as well as initial simulation results were also shown in [3]. Acting partners (c.f. fig. 2) can decide which energy system variant meets their predefined requirements best with respect to affordability and sustainability. The chosen energy system variant will be built in a second project part beginning in the middle of 2014.

## References

- [1] Schwan, T., Unger, R., Lerche, Dr.-Ing. C.: 1. Zwischenbericht – Erstellung eines integrativen Quartierskonzepts für das Quartier Schloß- und Ringstraße in Lohmen. EA Systems Dresden GmbH, June, 2013.
- [2] Unger, R.; Schwan, T.; Mikoleit, B.; Bäker, B.: Bessere regenerative Energieversorgung für Quartiere und Gemeinden durch systemübergreifende Simulation, 5. Internationaler Kongress Bauhaus SOLAR, Erfurt, 13./14. November 2012.
- [3] Schwan, T., Unger, R., Lerche, Dr.-Ing. C.: Model-Based Design of Integrative Energy Concepts for Municipalities, 16th ITI Symposium, Dresden, 12./13. November 2013.
- [4] Schwan, T., Unger, R., Baeker, B., Mikoleit, B., Kehrer, C.: Optimization-Tool for local renewable energy usage in the connected system: Building-eMobility; 8th International Modelica Conference, Dresden, 20.-22. March 2011.
- [5] Schwan, T., Unger, R., Baeker, B., Mikoleit, B., Kehrer, C.: Optimization of local renewable energy systems using automotive simulation approaches, 12th Conference of International Building Performance Simulation Association, Sydney, 14.-16. November 2011.
- [6] Schwan, T., Unger, R., Bäker, B., Mikoleit, B., Kehrer, C., Rodemann, T.: "Green Building" - Modelling renewable building energy systems and electric mobility concepts using Modelica, 9th International Modelica Conference, München, 03.-05. September 2012.
- [7] K.-H. Dahlem, H. Heinrich: Einfluss des Grundwassers auf den Wärmeverlust beheizter Keller. 10. Bauklimatisches Symposium, Dresden, 1999.

# Enhancement of the building simulation software TRNSYS by coupling to the VEPZO model programmed in Modelica

Svea Kübler<sup>1,2</sup>

Victor Norrefeldt<sup>1</sup>

1: Fraunhofer-Institute for Building Physics, Fraunhoferstr. 10, D-83626 Valley

2: Transsolar Energietechnik GmbH, Curiestr. 2, D-70563 Stuttgart

[svea.kuebler@ibp.fraunhofer.de](mailto:svea.kuebler@ibp.fraunhofer.de) [victor.norrefeldt@ibp.fraunhofer.de](mailto:victor.norrefeldt@ibp.fraunhofer.de)

## Abstract

In this study, the possibility to interface a commercial building simulation tool with Modelica models is investigated. In this application, the zonal model VEPZO – modeled in Modelica – is coupled to the software TRNSYS – mainly programmed in Fortran – to be able to perform a dynamic co-simulation. The objective of this coupling is to obtain refined airflow and air temperature prediction, while retaining computation effort low enough to allow for transient computation. In a first attempt, a coupling using FMI was tested without success due to a lack of adequate solvers for FMI export. Therefore, a script coupling was implemented. Further steps include a validation and evaluation of the programmed interface and the results of the coupled system in respect to computation time, quality of results, usability and further development.

*Keywords: building simulation; zonal model; FMU, Dymola scripting, TRNSYS*

## 1 Introduction

Building simulation tools are used to assess building energy consumption, building control strategies, system performances, etc. on an annual or longer basis. In Modelica, the Buildings Library [1] and the Indoor Climate Library [2] contain models for the simulation of building energy performance. These libraries allow the simulation of building models directly in Modelica. Beside these, various commercial and free non-Modelica building simulation software products are available on the market. User's experience, knowledge of a specific tool and availability of tools in companies and institutes determine which

software is used. Therefore coupling of different tools becomes interesting when a specific model reaches its limits. In the presented study, an interface has been established for TRNSYS and the VEPZO model [3; 4] (programmed in Modelica) in order to enhance the simulation of atria.

### 1.1 TRNSYS

TRNSYS (Transient Systems Simulation) development started at the University of Wisconsin in 1975. The latest version 17 was released in 2010. TRNSYS code has a modular approach and is structured into types. Each type contains code for a specific task, usually representing a model of a specific component such as a part of an HVAC system. Types have a common structure and communicate via a given set of inputs, outputs and parameters. A central type for building performance simulations is the multizone building simulation environment (Type 56). It subdivides a building into multiple zones with homogeneous properties and conditions. The computation includes heat flows due to radiation, convection, conduction and solar gains and enthalpy flows through ventilation and air leakages. The integrated code COMIS [5] allows the formation of an airflow network by interconnecting zones. A schematic setup of the physical phenomena in an atrium is shown in Figure 1.

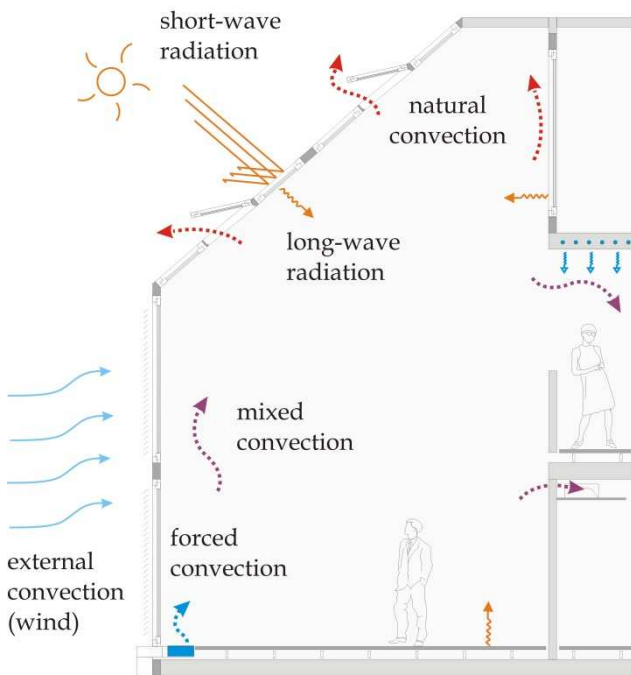


Figure 1: Schematic atrium model

The assumption of a perfectly mixed zone reaches its limits when simulating atria, because solar heat gains through glazing and a considerable height lead to thermal stratification of air. The current solution method to simulate atria in TRNSYS is to subzone them. For this, several air nodes are created in the atrium connected to one another by a predefined mass flow. To set up the model properly, the modeler requires prior knowledge of the airflow pattern – either from experience or from CFD simulations. However, even experienced modelers may fail to provide the proper airflow pattern. Due to a high requirement in computational power, CFD simulations are currently applicable for only a few configurations of different boundary conditions and only under steady-state conditions. Therefore, the extension of TRNSYS by a simpler airflow model is of interest.

## 1.2 VEPZO

The velocity propagating zonal model (VEPZO) was developed to fill the gap between the computationally cheap but inaccurate assumption of perfectly mixed air volumes and the slow but accurate CFD computation. Typically, a space is subdivided into  $10^1$  to  $10^2$  zones exchanging air [6]. The main components are a zone and a flow model which are connected by ports (Figure 2). Zone models possess a heat port to exchange convective heat flows from walls, equipment, humans, etc. In the zone model, conservation of energy and mass are implemented.

The flow model computes the airflow between adjacent zones resulting from pressure, momentum and height differences and losses. Velocity information is propagated through the models allowing the prediction of temperature and velocity distributions without prior knowledge of the airflow pattern. Models from the Modelica.Media package are used to compute air properties. Dymola is used as compiler for VEPZO. Results of the VEPZO model are the airflow pattern and temperature distribution in the considered space. Hence, it is suited to give a quick estimation of flow and air temperature conditions in an atrium.

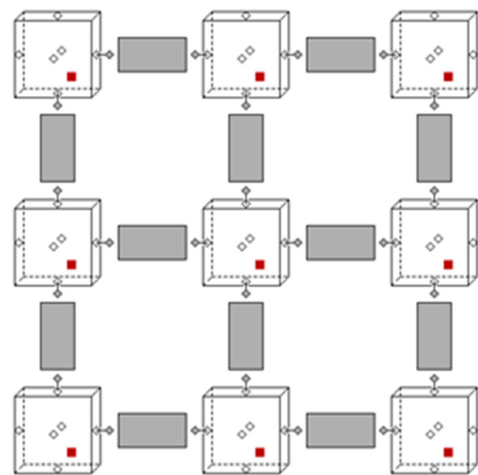


Figure 2: VEPZO model in x-z direction (y not shown); cubes: zones; grey rectangles: flows; rhombs: airflow ports; red solid squares: heat ports.

## 1.3 Model coupling

TRNSYS and VEPZO are coupled to benefit from the respective strengths of the codes. The goal is to obtain a correct simulation of the energetic performance and climatic conditions in an atrium without prior knowledge of the airflow pattern.

TRNSYS has already been used for co-simulation with other tools such as ESP-r, Energy-Plus and Matlab [7; 8]. All couplings use TRNSYS as a master and exchange data through a TRNSYS type which translates data into the required format.

For the coupling of Modelica models to other simulation environments several possibilities exist. A powerful tool is the Building Control Virtual Test Bed (BCVTB), an interface for building simulations developed by the Lawrence Berkeley National Laboratory. It already provides an interface for Modelica and TRNSYS. However, the coupling currently does not allow for iterations between components

[9]. Therefore, it is not suitable for a dynamic co-simulation as envisaged in this application.

In recent years the FMI-standard has been established for model exchange with other software environments [10]. It has already found several applications in building simulation technology [11] and is especially suited for Modelica models. Another possibility to couple other software to Modelica (Dymola) is the use of script files in the .mos-format which are automatically created and whose execution is launched in the command processor.

## 2 Method

TRNSYS and VEPZO communicate at the inner wall surface of the investigated atrium where TRNSYS passes the wall surface temperature to VEPZO. With this information, VEPZO computes the heat flow rate from the wall to the adjacent zone and hence resulting airflow and air temperature distributions. The heat flow rate is returned to TRNSYS to compute the energy balance (Figure 3). This coupling is iterated until the total energy balance converges below a defined threshold.

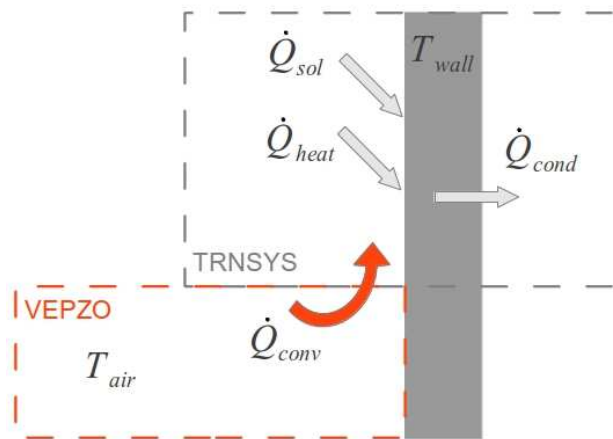


Figure 3: Heat flows and temperatures computed by the two interfaced tools

Both TRNSYS and the Modelica model VEPZO required some preparation before being interfaced. TRNSYS air nodes, which are required in the simulation environment, are inactivated by setting convective heat transfer coefficients to zero. The convective heat flux is then replaced by the one obtained from VEPZO.

VEPZO required TRNSYS convective heat transfer correlations to be implemented in the model. Input and output connectors were included to receive the surface air temperature and to deliver the heat flow.

Furthermore, supply airflow rate and temperature as well as exhaust opening pressure were set as inputs.

### 2.1 Model coupling via FMI

First attempts to simulate VEPZO as an FMU showed that in spite of having set input values for air temperatures and pressures, the FMU instantiates with all inputs being zero. This generates an error in the air model from Modelica.Media as 0 K and 0 Pa are out of definition bounds. Therefore, temperature and pressure offsets were introduced in the model (273.15 K, 101325 Pa, Figure 4). Thus, the temperature input is in °C and the pressure input gives the deviation from standard pressure.

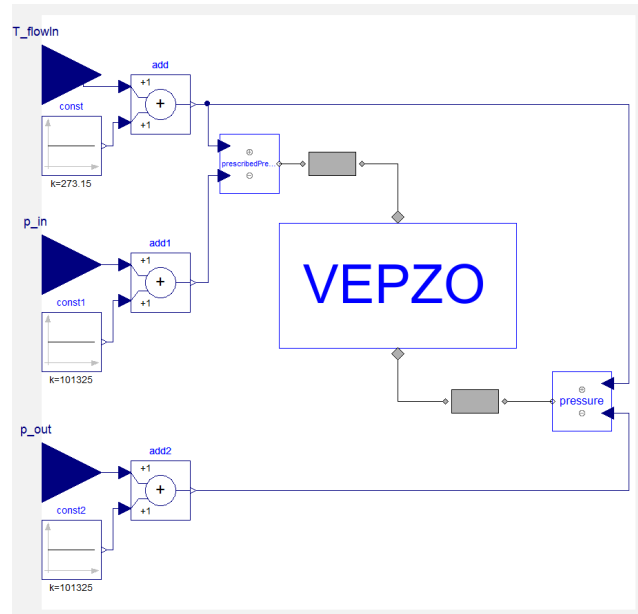


Figure 4: Introduction of temperature and pressure offset prior to FMU export

To allow the import of a VEPZO-FMU to TRNSYS, the new TRNSYS type was implemented (Figure 5). At the very first call the new type checks that the given FMU is suitable for co-simulation, retrieves the reference values of input and output parameters of the model and opens a result file which can be continuously accessed during the simulation. Then, the following steps are iterated:

1. Receive input values and current time as well as time step size from TRNSYS
2. Load the VEPZO-FMU
3. Instantiate VEPZO-FMU
4. Load values from the previous time step from internal storage
5. Initialize VEPZO-FMU with values from the previous time step

6. Perform calculation from the current to the next time step
7. Write results
8. Pass output values to TRNSYS.

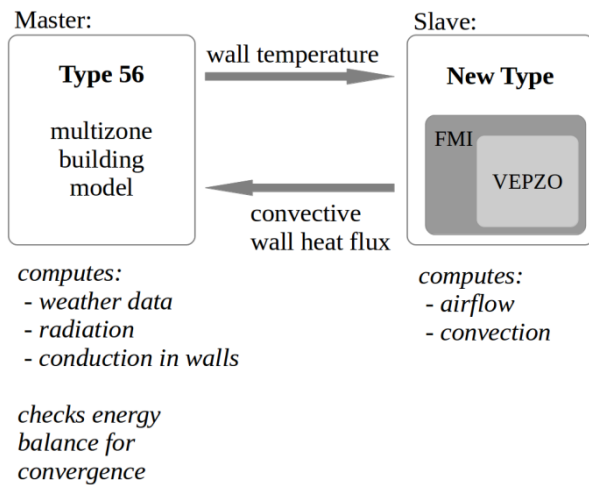


Figure 5: Basic structure of the two coupled tools in the interface via FMI

The coupling was implemented and simulation time showed to be significantly prolonged in comparison to VEPZO simulations in Dymola. When simulating a simple test case with a supply mass flow rate of 0.1 kg/s, supply temperature of 10 °C and wall temperature of 30 °C the air temperature predicted by the FMU showed an unstable range between 180 s and 500 s simulation time in comparison to the simulation in Dymola (Figure 6). The reason for this is that VEPZO requires a fifth-order stiff Runge-Kutta solver in order to be solved for low supply mass flow rates. This solver is not available for export in the current version of Dymola, but will be implemented in future versions [12]. Hence the dynamic coupling of TRNSYS and VEPZO via FMU was discarded in this study.

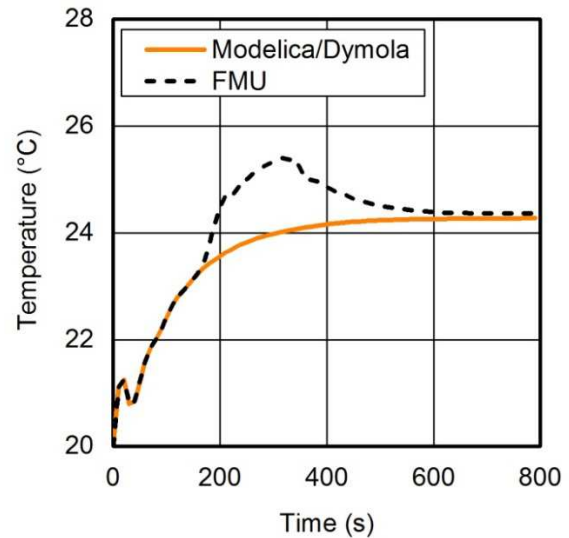


Figure 6: Comparison of Dymola and FMU simulation of a simple test case in VEPZO

## 2.2 Model coupling through scripting

In a second approach TRNSYS and VEPZO were coupled using Dymola scripting. For this, another new TRNSYS Type was implemented. This type writes and executes a .mos-file containing wall temperatures, simulation settings and code for export of computed wall heat flow rates into a .csv file. The new type reads the .csv file and uses it as input for the thermal building simulation (Figure 7).

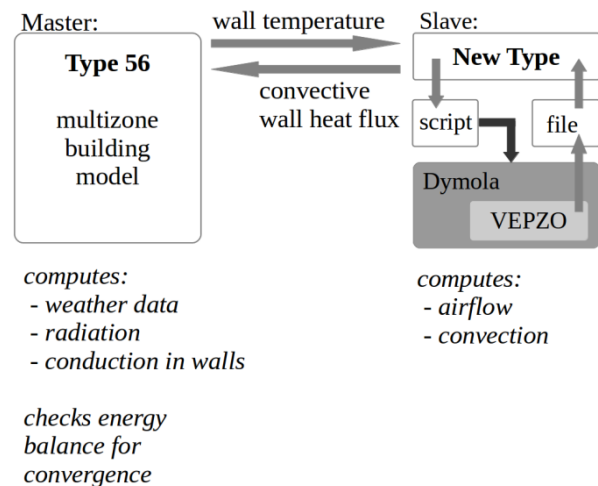


Figure 7: Basic structure of the two coupled tools in the interface via scripting

### 3 Atrium model

To demonstrate the coupled simulation of TRNSYS and VEPZO an atrium is modeled and simulation results are compared to data from measurements and CFD simulations. The modeled atrium (Figure 8) is part of the Concordia University engineering building located in downtown Montreal (45.5°N, 74°W). In total, five atria each covering three floors are placed on top of each other. The considered atrium is of rectangular geometry and can be conditioned by both natural and mechanical ventilation. Mouriki [13] conducted measurements in this atrium at four days ranging from August to November 2007. Based on this data, Hussain [14] carried out several CFD simulations of this atrium with different turbulence models.



Figure 8: Considered atrium

To reduce the uncertainty of boundary conditions, a day with only mechanical ventilation (November 2<sup>nd</sup> 2007) has been selected for comparison. It was a clear and cold day with ambient temperatures between -1.5 and 8 °C. The maximum global horizontal radiation for a nearby weather station was 480 W/m<sup>2</sup> [15]. Measurement readings are available from 6am to midnight. Figure 9 shows the geometry of the considered atrium and the location of mechanical ventilation inlet and outlet. The atrium air volume is divided into 5 x 5 x 6 (length, depth, height)

VEPZO zones. To compare the coupled simulation to the classical TRNSYS approach using modeler's experience, a second simulation model using only TRNSYS was set up. In this model, air is subdivided into three horizontal slices. Walls are discretized into a total of 20 facets in order to account for different wall temperatures due to varying solar radiation. For this work, we could not access construction plans of the atrium. The only hint to guess materials are published pictures and transient temperature measurements. Therefore, this information presents a considerable source of error which is mainly reflected in the thermal capacity of the system.

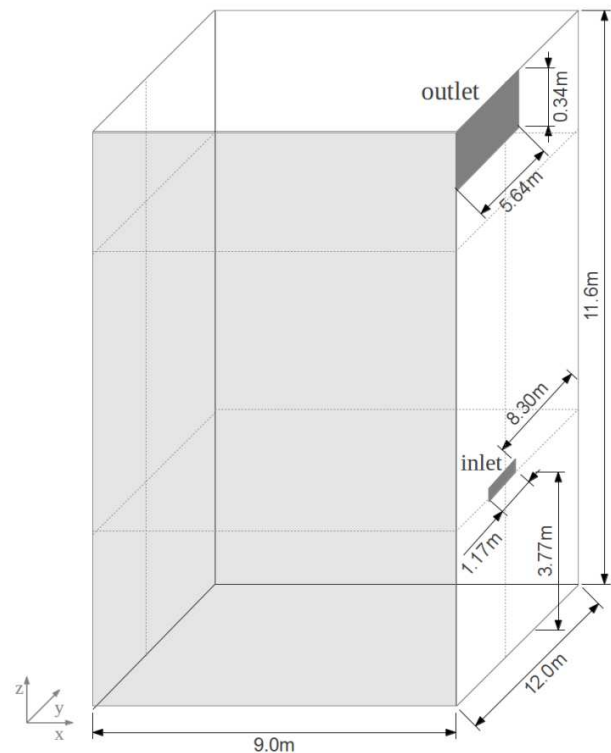


Figure 9: Geometry of considered atrium

The boundary conditions obtained from measurements are inner glazing surface temperatures, transmitted solar radiation, ambient temperature and supply air conditions (Figure 10). To acquire a proper initialization of wall temperatures, four days prior to the considered day were simulated before the coupled simulation was started. The coupled simulation took 4hr and 45min.

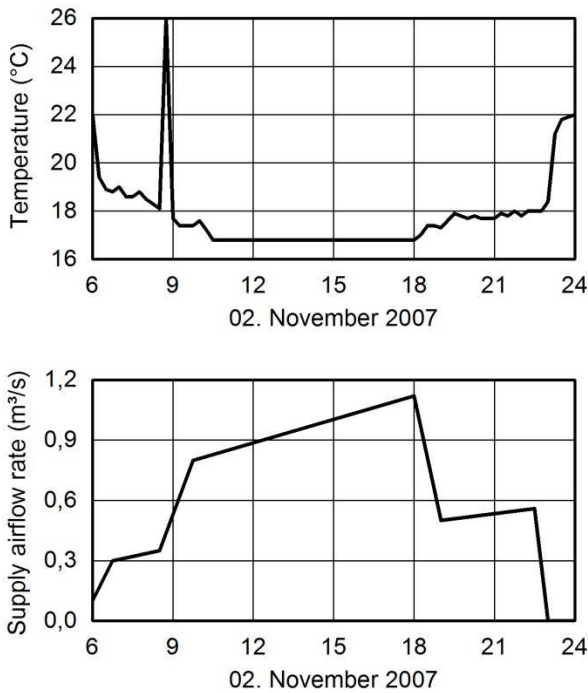


Figure 10: Supply air boundary conditions

Figure 11 shows the comparison of measurement and coupled simulation for the average temperature in three heights of the atrium. Five main observations are made:

- The results from the coupled simulation show an initial drop in air temperature which causes the further temperature profiles to be lower than measured.
- The peak in temperature is reached by about one hour delay in the simulation but shows good coherence in the absolute value with the measured peaks.
- The model is more sensitive to a change in airflow boundary condition than measurements show.
- A raise in temperature due to a lower cooling flow rate around 6.30pm can only be observed at the lowest measurement position but is observed for all heights in the simulation. The same is the case for the shut-down of the ventilation around 11pm.
- As a result, the air temperature at the end of the simulation is about 2 K higher than measurement shows.

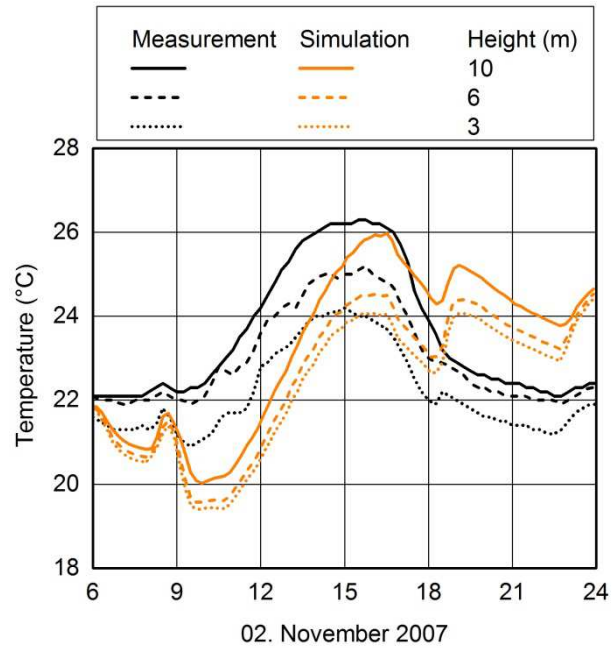


Figure 11: Temperature at three different heights over the course of one day

Figure 12 shows a comparison of measured air temperature profile, coupled simulation results and CFD simulations [14] for 4pm. The coupled simulation shows considerably better agreement to measurements than CFD simulations do.

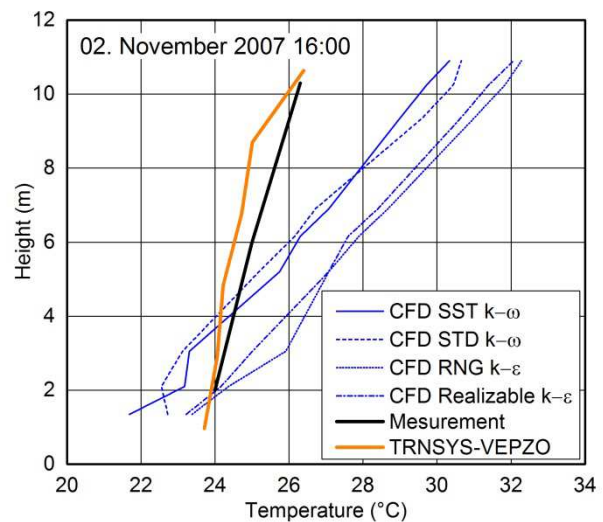


Figure 12: Comparison of simulated and measured temperatures over the height at 4pm



## 4 Discussion

The higher rate of response of air temperature of the model to variations in air supply suggests that it does not contain enough thermal capacities dampening variations of boundary conditions or that the convective heat coefficient model is inaccurate. Additional capacities can for example be caused by furniture and other internal objects like stairs. The initial drop in temperature implies that the surface's temperatures are lower in the simulation than they are in reality. This could be caused by a lower thermal mass of walls in the model or because the atrium gains energy from surrounding rooms during the night.

Between 3 and 4pm relatively stable temperature conditions are present in the atrium. For this case, the coupled simulation gives better results than CFD computations. The reason is that it is very difficult to obtain a good prediction of airflow by CFD for such large volumes with comparatively small openings without an adequate benchmark case.

The results from the dynamically coupled simulation in comparison to those obtained from a TRNSYS simulation show that very similar temperature profiles can be achieved for simple configurations where the airflow pattern can be predicted by experience. This implies that VEPZO results show a high dependence on the inputs obtained from TRNSYS.

## 5 Conclusions and future work

A dynamic coupling of the building simulation tool TRNSYS and the Modelica model VEPZO has been implemented successfully using Dymola scripting. A coupling through FMI was not successful due to missing solvers for FMI for co-simulation export.

The results obtained from the dynamically coupled simulation show significant dependence on the TRNSYS model, especially on the proper knowledge and implementation of the building construction (materials, capacities, etc.). If those are modeled accurately, coupled simulations can give good predictions of airflow and air temperature as required in the building planning process. In comparison to CFD, results of the coupled simulation show better coherence with measured data.

The usability of the current coupling method should be improved by speeding up computation times and facilitating geometry specific scripting. It could be increased by using FMI once the required solvers are available for export. It would be helpful if FMI instantiation could be performed at values different

from zero to avoid error-prone preparation of the Modelica model.

Further development of VEPZO as deduced from this study includes the implementation of heat transfer correlations which take into account the wall orientation and flow velocity. Additionally the feature of an artificial internal heat capacity will be integrated in order to be able to model furniture.

Regarding the interface, a possible further development is the integration of the multi-zonal model COMIS into the interface such that boundary conditions at atrium openings can be obtained from a coupled COMIS-TRNSYS simulation. Once the computation time of the coupling is reduced, the interface could moreover be utilized to optimize the design of atriums and large halls such as the best location and size of ventilation openings.

## References

- [1] Wetter, M.: Modelica library for building heating, ventilation and air-conditioning systems, 7th International Modelica Conference, Como, Italy, 20.-22. september 2009
- [2] Norrefeldt, V., Andersson, D., Pathak, A. et al.: The Indoor Climate Library and its application to heat and moisture transfer in a vehicle cabin, 9th Modelica Conference, 3.-5. september 2012, Munich, Germany
- [3] Norrefeldt, V., Grün, G., Sedlbauer, K.: VEPZO - Velocity propagating zonal model for the estimation of the airflow pattern and temperature distribution in a confined space, Building and Environment, Volume 48, pp. 183-194, 2012
- [4] Norrefeldt, V., Grün, G.: VEPZO - Velocity Propagating Zonal Model for the prediction of airflow pattern and temperature distribution in enclosed spaces, 9th Modelica Conference, 3.-5. september 2012, Munich, Germany
- [5] LBNL, consulted 7-3-2012, <http://epb.lbl.gov/comis/>
- [6] Boukhris, Y., Gharbi, L., Ghrab-Morcous, N.: Modeling coupled heat transfer and air flow in a partitioned building with a zonal model: application to the winter thermal comfort, Building Simulation, Volume 2, pp. 67-74, 2009
- [7] Beausoleil-Morrison, I., Macdonalda, F., Kummert, M. et al.: The design of an ESP-r and TRNSYS cosimulator, Building

- Simulation, 14.-16.november 2011, Sydney, Australia
- [8] Trcka, M., Wetter, M., Hensen, J.: Comparison of co-simulation approaches for building simulation and HVAC/R system simulation, Building Simulation, 3.-5. september 2007, Beijing, China
  - [9] Wetter, M.: Building Controls Virtual Test Bed UserManual, 2012
  - [10] Modelica Association: Functional Mock-up Interface, 2012
  - [11] Pazold, M., Burhenne, S., Radon, J. et al.: Integration of Modelica models into an existing simulation software using FMI for Co-Simulation, 9th Modelica Conference, 3.-5. september 2012, Munich, Germany
  - [12] Wernersson, K.: Personal Communication, 2013
  - [13] Mouriki, E.: Solar-Assisted Hybrid Ventilation in an Institutional Building, Master Thesis at Concordia Universtiy Montréal, Canada, 2009
  - [14] Hussain, S.: Numerical investigations of the indoor thermal environment in atria and of the buoyancy-driven ventilation in a simple atriumbuilding, PhD Thesis at the Queen's University, Montréal, Canada, 2012
  - [15] National Renewable Energy Laboratory: NSRDB - National Solar Radiation Data Base, United States, 2007

# The Modelica Thermal Model Generation Tool for Automated Creation of a Coupled Airflow, Radiation Model and Wall Model in Modelica

Arnav Pathak, Victor Norrefeldt, Abdellah Lemouedda, Gunnar Grün  
Fraunhofer Institute for Building Physics, Dept. Indoor Climate, 83626 Valley, Germany  
[arnav.pathak@ibp.fraunhofer.de](mailto:arnav.pathak@ibp.fraunhofer.de), [victor.norrefeldt@ibp.fraunhofer.de](mailto:victor.norrefeldt@ibp.fraunhofer.de),  
[abdellah.lemouedda@ibp.fraunhofer.de](mailto:abdellah.lemouedda@ibp.fraunhofer.de), [gunnar.gruen@ibp.fraunhofer.de](mailto:gunnar.gruen@ibp.fraunhofer.de)

## Abstract

This paper presents the Modelica Thermal Model Generation Tool. The aim of this tool is to enable the user to set up a geometrically correct thermal model for complex geometries that allows predicting the impact of heated/heating devices and their location both in terms of airflow pattern and radiation distribution. Using a geometry file exported from CAD software, the tool distributes wall facets, air nodes and computes the long-wave radiant view factor matrix for obstructed and unobstructed surfaces. This information is exported as ready to use Modelica code. The zonal model VEPZO is used to model airflow within a domain (enclosed space). This model allows predicting airflow and air temperature distribution in space on a coarse mesh and thus computes faster than classical CFD computations. Walls are subdivided on the same grid as the zonal model is set upon. For each wall facet, the Modelica Thermal Model Generation Tool computes the view factors to the other facets in the domain.

Comparison of simulated results with test data and application of the Modelica Thermal Model Generation Tool for a room with radiant heating and for the cooling of an aircraft cockpit are presented in this paper.

*Keywords: Thermal Model, VEPZO, Airflow Simulation, View Factor Calculation, Long-wave Radiant Heat Exchange, LowRad*

## 1 Introduction

A number of codes for thermal simulation (e.g. Indoor Climate Library [1], Buildings Library [2]) have emerged in Modelica. These models are set up in a way to allow the user a comfortable parameterization with a moderate number of models. The resolution of these models is on room level. A

typical model would consist of six rectangular enclosures, a perfectly mixed air volume and a radiation node estimating the view factors in a simplified way. Connecting a number of such basic elements and modelling an airflow network between rooms (e.g. the Multizone model of Buildings Library [3]) yields a building. To refine the airflow and temperature distribution within a room, zonal models have been implemented in Modelica ([1; 4]). The manual setup of these models is easy when only rectangular spaces are considered. Simple non-rectangular spaces with a low number of inclined surfaces can still be set up manually, too. However, the manual setup of a thermal model for more complex geometries becomes increasingly time consuming and error prone with increasing “non-rectangularity” of the geometry as surfaces and zones require a one-by-one parameterization of area, volume and location. Furthermore, the simplified radiation approach distributing radiation proportionally on surfaces (e.g. described in [5]) is only validated for building applications where typically temperature differences between surfaces are relatively small. In other configurations, the simplified approach may result in high errors of surface temperatures due to wrong estimation of view factors, which can further cause error in energy balance and estimation of comfort level.

One of the purposes of creating the Modelica Thermal Model Generation Tool is to allow estimating the impact of the location of a very hot device or the development of local radiant or convective heating strategies.

To achieve this goal, the Modelica Thermal Model Generation Tool combines two Modelica codes presented at the Modelica Conference 2012. The VEPZO model [6] computes the airflow and temperature distribution in space. Up to now, it has only been used on rectangular geometries. For non-rectangular spaces, the geometry was manually approximated by rectangular elements. The Modelica Long-wave Radiation View Factor Model [7] allows

for view factor computation based on actual geometries and creates a more detailed radiation node model. The long-wave radiation view factor model presented in Modelica Conference 2012, was not considering the obstruction check for obstructed views.

## 2 Method

This section outlines the steps of the Modelica Thermal Model Generation Tool and used models.

### 2.1 Modelica Thermal Model Generation Tool

The Modelica Thermal Model Generation Tool (MThMGT) is a pre-processing tool developed in C++ language. It translates a CAD model exported into the .stl format to Modelica code that can be simulated in the Modelica simulation environment. The .stl format describes the geometry by a series of triangles and their normals. When opening the MThMGT GUI (Figure 1), the user selects the input .stl-file and determines a grid for the subdivision of the geometry into zones. The user can add heat sources and ventilation openings in the GUI.

Based on the user-defined grid, the MThMGT distributes nodes and checks for each node whether it is within or outside the geometry. Furthermore, it checks whether two adjacent nodes are separated by a wall indicating that they belong to two different domains (i.e. room1 and room2). The nodes within one domain form the edges of the zones. The “classical” rectangular zone is thus determined by eight nodes. Zones adjacent to the geometry are defined by less than eight nodes as some nodes are outside the domain. For these zones, dedicated rules are applied approximating zone geometry and creating wall facets that approximate the actual geometry. Each zone is given a unique name to be identified for its connections to walls, heat sources, convection and radiation models and ventilation openings. To enable the user to assess the created thermal model, the geometry of created zones and wall facets is saved as .stl file in dedicated repositories. Figure 2 shows an example for the distribution of zones in geometry.

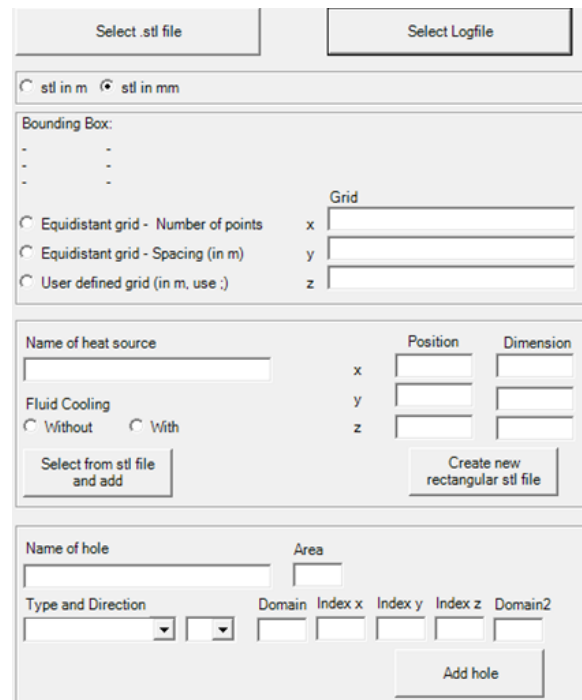


Figure 1: GUI of the MThMGT - Zonal Part

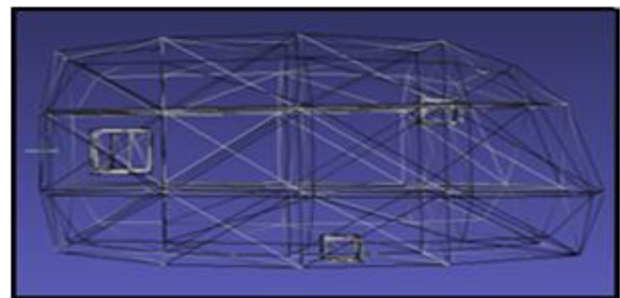
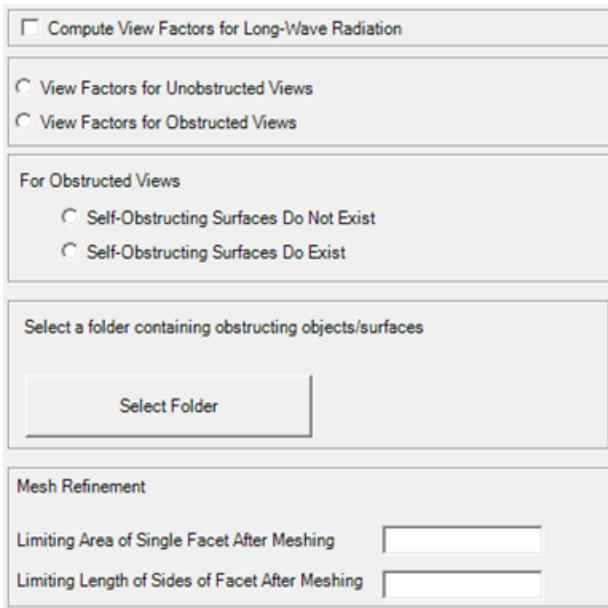


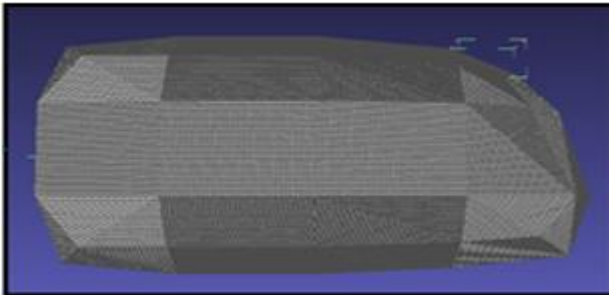
Figure 2: Example for created zonal grid

View factor computation settings are entered in the second part of the GUI of MThMGT (Figure 3). First, the user needs to state whether view factors shall be computed and what kind the geometry is of. Unobstructed view means all the surfaces in one domain have complete view of each other if they are not in the same plane. If there is any obstruction in the view of one surface to another surface, the user must select computation method for obstructed views. Similarly, if the surface is self obstructed, for example a surface is shaped as an alphabate ‘S’, the user must select the computation method for self-obstructed views. The aim of these selection possibilities is to obtain a quicker computation by avoiding unnecessary obstruction checks.



**Figure 3: GUI of the MThMGT - Radiation Part**

Accuracy of view factor computation is directly proportional to the mesh quality of a surface. Therefore, mesh refinement is required. For this, the user can define a limiting area and a limiting length of facet edges for refined triangles. This ensures that all facets are refined below this threshold. Figure 4 shows a refined mesh of the geometry shown in Figure 2.



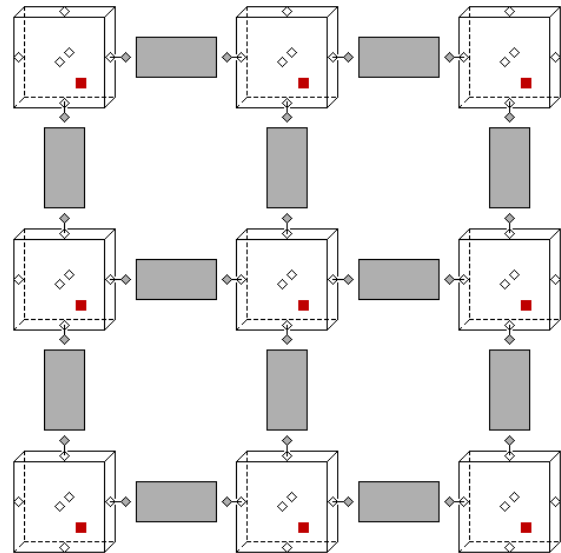
**Figure 4: Refined meshing of created geometry for view factor computation.**

After computation of zonal grid and view factors, the MThMGT does all the appropriate connections between the zonal model, the radiation model, the convection models and the wall models. It includes heat sources and ventilation openings to the model and connects them with the appropriate zone and radiation node. The MThMGT writes all the model descriptions and connections into a Modelica file (.mo file) which can be further used in a Modelica environment for thermal simulation.

## 2.2 VEPZO zonal airflow model

The two main components of the VEPZO model are a zone model and a flow model (Figure 5). The zone (cube) and the flow (grey rectangle) models are connected by ports to form a room. These ports allow the exchange of relevant information between the flow and the zone model. The flow models have two ports to connect adjacent zones. Each zone has six ports, one for each boundary. A boolean parameter is assigned to each port to make the distinction whether the port is connected to a flow model or whether there is no flow because the zone is adjacent to a room boundary surface. Furthermore, each zone has a heat port (red square) allowing heat exchanges with models of other components like heat sources or walls.

With the MThMGT, these connection parameters are automatically set and ports are automatically connected to the respective models.



**Figure 5: Zonal model in x-z direction (y not shown); cubes: zones; grey rectangles: flows; rhombs: airflow ports; red solid squares: heat ports.**

The main task of the zone model is to compute mass and energy balances and air properties (density, enthalpy, pressure, temperature, etc) using air models of Modelica.Media.

$$\frac{\partial \rho_i}{\partial t} = \frac{\sum_j \dot{m}_{i,j} + \sum_{\text{sources}} \dot{m}_{\text{source},i}}{V_i} \quad (1)$$

$$V_i \cdot \rho_i \cdot \frac{\partial h_i}{\partial t} = \sum_j \dot{m}_{i,j} \cdot h_{i/j} + \sum_{\text{sources}} \dot{m}_{\text{source}} \cdot h_{i/\text{source}} + \sum_{\text{heatflows}} \dot{Q} \quad (2)$$

Where pressure is constant,  $V$  is the volume,  $\rho$  is the density,  $h$  is the enthalpy,  $m$  are entering and leaving airflows, and  $Q$  are heat flows in the zone.

The main task of the flow model is to compute the airflow rate between two adjacent zones. For this, forces resulting from pressure, momentum and height difference and losses are summed yielding the acceleration of the airflow in the flow paths. A detailed set of equations for the flow path can be found in [6].

### 2.3 Long-wave Radiant Heat Exchange Model: LowRad

The radiant heat exchange of a surface depends on the temperature  $T$ , the reflectivity  $\rho$ , the emissivity/absorptivity  $\varepsilon$  and for transparent surfaces the transmissivity  $\tau$ .

For an opaque surface  $i$  the incoming ( $Q_{i,in}$ ) and leaving ( $Q_{i,out}$ ) radiant heat flows are computed by:

$$Q_{i,out} = A_i \cdot \varepsilon_i \cdot \sigma \cdot T_i^4 + \rho_i \cdot Q_{i,in} \quad (3)$$

$$Q_{i,in} = \sum_j F_{i-j} \cdot Q_{j,out} \quad (4)$$

with  $\sigma$ : Boltzmann number ( $5.67 \cdot 10^{-8} \text{ W/m}^2\text{K}^4$ ),  $F_{i-j}$ : view factor of surface  $j$  to  $i$ .

Determination of the long-wave radiant heat exchange between surfaces requires the view factor matrix  $F_{i-j}$ . There are several analytical solutions available to calculate view factors for simple and known configurations. Many building simulation programs estimate the view factors in a simplified way, especially when complex geometries are involved. The simplified approach may result in high errors of surface temperatures, which can further cause error in energy balance. Hence it is important to determine accurate view factors. The purpose of creating this model is to calculate view factors between complex geometries for unobstructed, obstructed and self-obstructed views.

#### 2.3.1 View factor matrix integral

To assess the radiative exchange between non-obstructed surfaces, the corresponding view factors

between the surfaces need to be computed. View factors between two surfaces are dependent on the geometry of the surfaces and their orientation. The view factor can be interpreted as the fraction of diffusive radiant heat exchange between surface  $i$  and surface  $j$ . The view factor between two surfaces is obtained from the integral over the view factors between the infinitesimal surface elements  $dA_i$  and  $dA_j$  (equation (5) and Figure 6):

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot r^2} dA_i dA_j \quad (5)$$

where  $r$  is the distance between the centres.

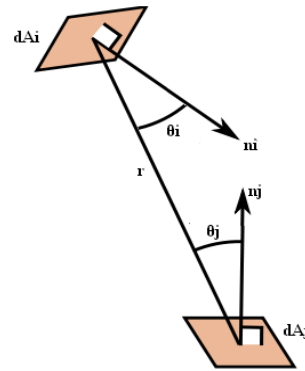


Figure 6: View factor between two surfaces  $i$  &  $j$

It is sufficient to compute the upper triangle of the view factor matrix. The lower triangle can be constructed by the following relationship:

$$A_i \cdot F_{ij} = A_j \cdot F_{ji} \quad (6)$$

#### 2.3.2 View factor matrix computation

The input for the computation of the view factor between two surfaces  $i$  and  $j$  is their triangulation in the .stl format. The MThMGT reads vertices and the normal vector of each triangular facet. The model calculates the centre and area of each triangle, the distance between each triangle and all the other triangles and similarly directional cosines for each triangle pair. For this, the discretization of equation (5) is used:

$$F_{ij} = \frac{1}{A_i} \sum \sum \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot r^2} \cdot dA_i dA_j \quad (7)$$

Required angles are determined by using following equations:

$$\cos\theta_i = \frac{l_i(x_j - x_i) + m_i(y_j - y_i) + n_i(z_j - z_i)}{r} \quad (8)$$

$$\cos\theta_j = \frac{l_j(x_i - x_j) + m_j(y_i - y_j) + n_j(z_i - z_j)}{r} \quad (9)$$

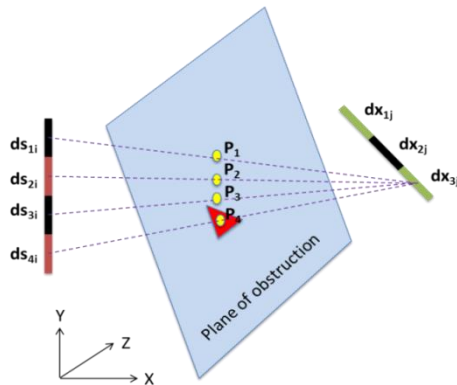
with  $l, m, n$ : directional components of the normals of  $dA_i, dA_j$

$x, y, z$ : coordinates of the centers of the surface elements.

Summing this discretized integral over all triangles forming the two surfaces yields the view factor between these two surfaces. Once the view factor  $F_{ij}$  from surface  $i$  to surface  $j$  is known, the reciprocal view factor from surface  $j$  to  $i$   $F_{ji}$  can be calculated from equation (6).

### 2.3.3 Obstruction check

To check whether two surface elements are obstructed by a third element, the plane of eventual obstruction needs to be set up.



**Figure 7: Obstructed view factor between surface  $i$  and surface  $j$**

The equation of plane is given by

$$A \cdot x + B \cdot y + C \cdot z + D = 0 \quad (10)$$

$$D = -(A \cdot x_{ob} + B \cdot y_{ob} + C \cdot z_{ob}) \quad (11)$$

Where  $x_{ob}, y_{ob}$  and  $z_{ob}$  are the vertices of the obstructing triangular facet (in this case only one facet is obstructing the view). The equation of the line connecting centers of facets on surface- $i$  (i.e. centers of  $ds_{1i}, ds_{2i}, ds_{3i}, ds_{4i}$ ) and surface- $j$  (i.e. centers of  $dx_{1j}, dx_{2j}, dx_{3j}$ ) checks the relative position between the line and the obstructing plane. There are three possibilities:

- The line can be parallel to the obstructing plane

- The line can belong to the obstructing plane
- The line can intersect the obstructing plane.

For this, the relative position of the obstructing surface to the line is checked:

$$m = \frac{A \cdot x + B \cdot y + C \cdot z + D}{A \cdot a + B \cdot b + C \cdot c} \quad (12)$$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x_j - x_i \\ y_j - y_i \\ z_j - z_i \end{pmatrix} \quad (13)$$

If the plane of obstruction is between surfaces  $i$  and  $j$ , the value of  $m$  is greater than 0 and less than 1.

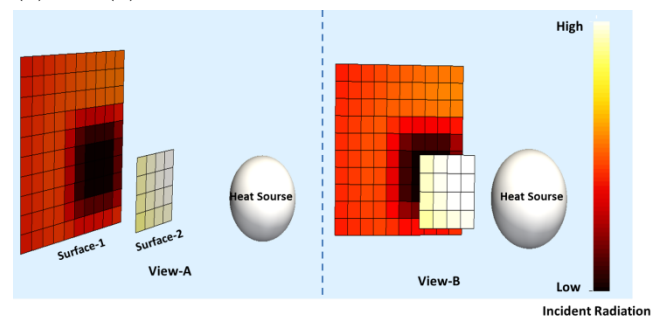
If the denominator of equation (12) is not zero, the point  $p$  of the intersection between the line and the plane can be found using:

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} x_i + a \cdot m \\ y_i + b \cdot m \\ z_i + c \cdot m \end{pmatrix} \quad (14)$$

Once the point  $p$  is known, the next step is to find out using barycentric technique if the point  $p$  is inside the obstructing triangle or not. If the point  $p$  is inside the triangular facet the view factor is assumed to be zero.

### 2.3.4 Radiation node model

The radiation node model calculates long-wave radiant heat exchange between  $n$  surfaces. It contains  $n$  thermal ports, surface properties and the view factor matrix. Each port connects to the corresponding surface model. The incoming and outgoing radiations for each surface are computed using the view factor matrix, temperature at each thermal port and surface properties as per equations (3) and (4).



**Figure 8: Incident radiation on two surfaces**

Application example of radiation model with obstructed view can be seen in Figure 8. There are two surfaces in front of radiant heat emitter (Globe).

Some part of surface-1 is obstructed by surface-2. Different colors represent difference in intensity of incident radiation. Due to the lower distance from the globe to surface-2, it has a higher incident radiation than surface-1. The view factor of the globe to the obstructed part of surface-1 is zero. Hence there is no direct radiant heating of the obstructed part.

### 2.4 Walls and convection models

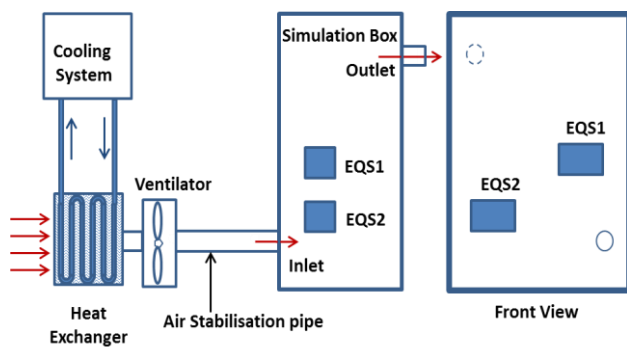
Convection models are used from the Modelica Standard Library. Wall models are based on a suite of thermal capacitances and thermal resistances from the Modelica Standard Library. The parameterization of these models yields the different materials that can be used.

The MThMGT exports the wall type for each domain with a “none” wall. This wall has no capacitance and no thermal resistance. In the exported code, the model type can be changed to the wall actually used in the geometry.

## 3 Coupled Model Validation

The coupled airflow, radiation and wall model generated with the MThMGT has been validated on a test bench.

### 3.1 Test Setup

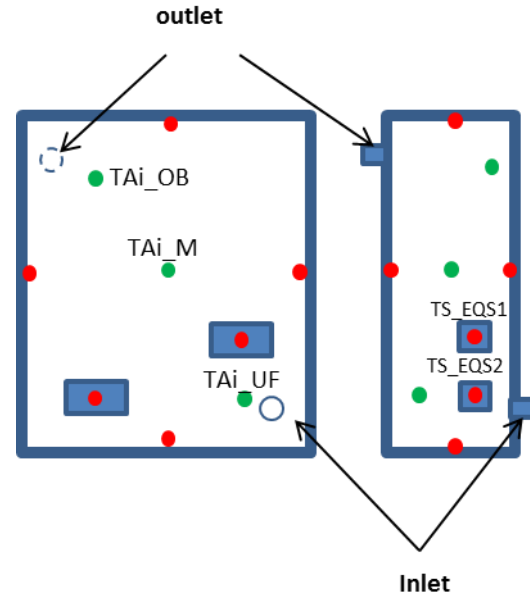


**Figure 9: Experimental Setup: Box with air conditioning system**

The experimental setup is composed of a box of size 2m x 1.5m x 0.9 m connected to an air supply system and insulated with 10 cm thick Armaflex foam. The box contains two heat sources in a form of equipment simulators (EQS) (Figure 9). These are installed in a test zone in a manner to have air circulation around them. By hanging them with a thin non-metallic rod their support can be considered

to be thermally insulated. Heating foils release a uniform heat flow on all surfaces of the EQS.

Conditioned air is supplied to the simulation box with a pipe of 104 mm diameter. In order to get a well developed flow at the inlet a minimum pipe length of  $L=15x D_{hydraulic}$  is used. This well developed flow is also required for a high inlet velocity measurement accuracy.



**Figure 10: Sensor positions: green dots: location of air temperature sensors, red dots: location of surface temperature sensors**

Figure 10 shows the distribution of measurement locations that were used for model validation in this study. Table 1 describes four different validation test cases varying supply airflow rate and long-wave emissivity of the EQS.

**Table 1: Test Cases (TC1-4), Al: aluminium surface, B: black painted surface**

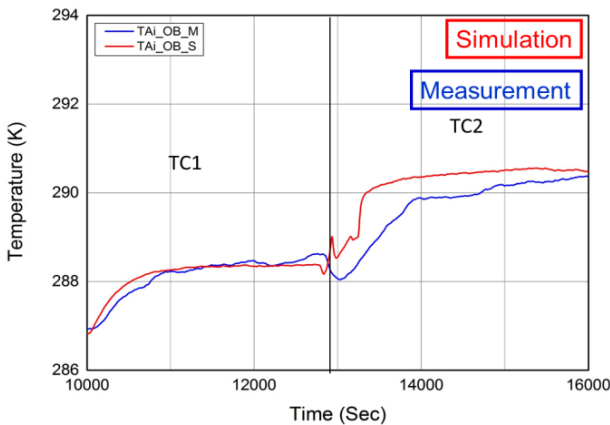
	TC1	TC2	TC3	TC4
Inlet mass flow (kg/min)	10	7.5	10	7.5
Supply Temperature (K)	283.15	283.15	283.15	283.15
Pressure (hPa)	940	940	940	940
Power EQS1 (W)	500	500	500	500
Power EQS2 (W)	500	500	500	500
Surface EQS1	Al	Al	B	B
Surface EQS2	Al	Al	Al	Al



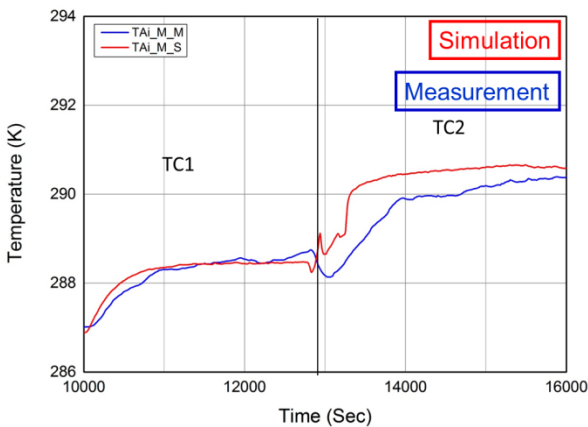
### 3.2 Results

To validate the coupled model, simulated results of four different test cases were compared with actual test results.

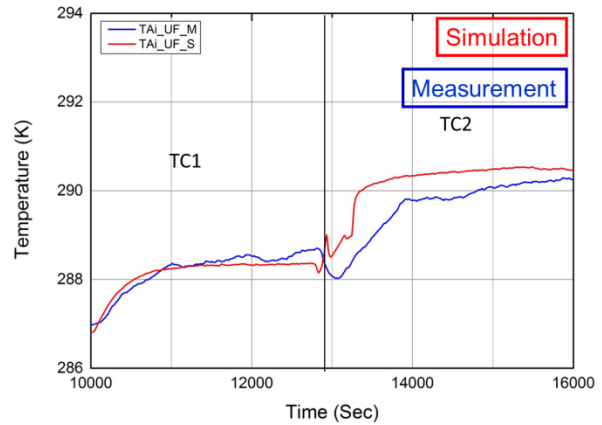
Figure 11 to Figure 13 show the comparison of simulated and measured air temperatures at the top left corner (TAi\_OB), the center (TAi\_M) and the bottom right corner of the box (TAi\_UF) for TC1 and TC2. In steady-state conditions, the model predicts air temperature with less than 0.5 K deviations.



**Figure 11: Air temperature at the top left corner of the box**

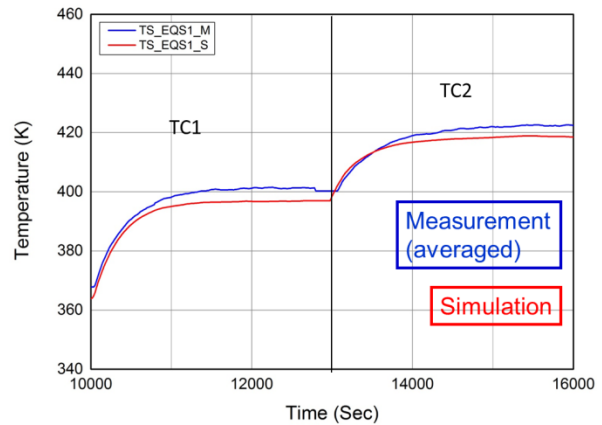


**Figure 12: Air temperature at the center of the box**

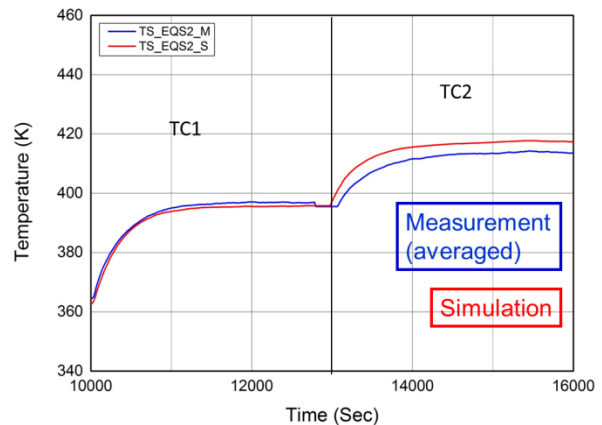


**Figure 13: Air temperature at the bottom right corner of the box**

Figure 14 and Figure 15 show the comparison of simulated and measured surface temperatures of EQS1 and EQS2. In the experimental setup, surface temperature sensors are placed on each face of the EQS. For model validation, these temperatures were averaged to obtain one representative surface temperature. At steady state, the deviation of measurement and simulation is less than 5 K.

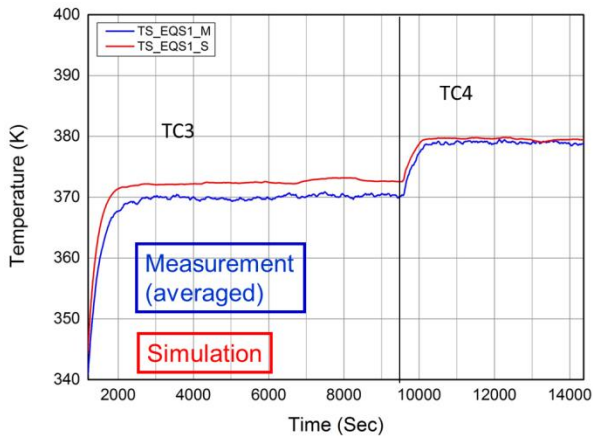


**Figure 14: Surface temperature of EQS1 for TC1 and TC2**

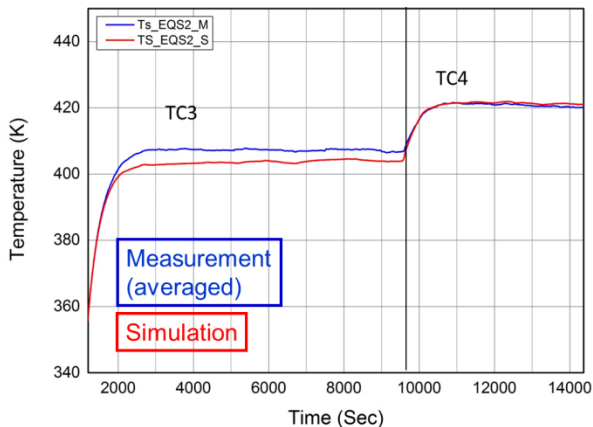


**Figure 15: Surface temperature of EQS2 for TC1 and TC2**

To assess model validity for changed long-wave radiation properties, EQS1 was painted black for TC3 and TC4. Figure 16 and Figure 17 show the comparison of simulated and measured surface temperatures of EQS1 and EQS2. Due to higher radiative exchange, surface temperature of EQS1 is lower than that of EQS1 in TC1 and TC2. EQS2 receives more radiation from EQS1 resulting in a higher surface temperature compared to TC1 and TC2. The accuracy of model predictions is in the range of 5 K for this setup, too.



**Figure 16: Surface temperature of EQS1 for TC3 and TC4**



**Figure 17: Surface temperature of EQS2 for TC3 and TC4**

### 3.3 Discussion

Comparison with validation test data shows that results of air temperatures show slight variations during transients but are accurate in steady state. The model predicts the surface temperature of EQS well for both transient and steady state conditions and for different emissivity. The impact of a higher radiative loss of the black EQS might have been visible even in a simplified radiation approach. However, the relatively higher surface temperature of the other

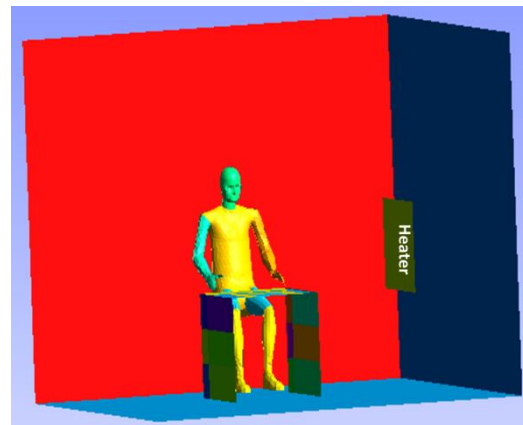
EQS, still being low emissive, would not have been correctly predicted without the computed view factor matrix finding a considerable radiative heat exchange between both EQS.

## 4 Application Examples

This section shows two application examples that have been setup using the MThMGT.

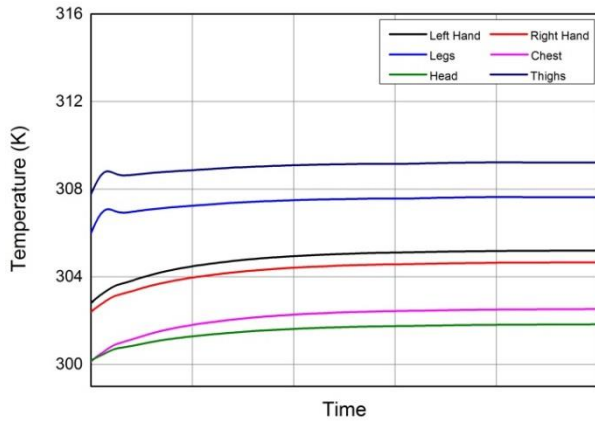
### 4.1 Radiant heating of a room

In this example the MThMGT has been used to assess the impact of radiant heating. Figure 18 shows three sides of a room with a wall heater, a table and a sitting manikin. The table is located in the center of the room. It has been subdivided into ten parts, three on the right, three on the left and four on the lower side of the desk. The impact of using these parts as radiant heating sources by applying a heating foil on them is investigated. Zoning has been done such that each heat source lies in a separate zone.



**Figure 18: MThMGT application example of building room**

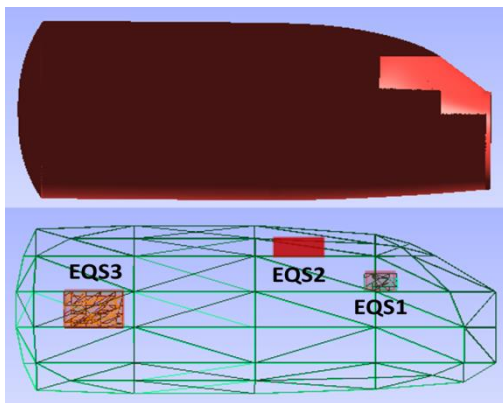
Figure 19 shows the result of different parts of the manikin. The temperatures of legs and thighs are higher than other body parts because of the radiant heating from the heating foils on the inner sides of the table. The temperature of the left hand is higher than that of the right hand, as the left hand is closer and in line of the wall heater. The left part of the chest is cooler than the left arm even though being in line with the heater due to the obstruction caused by left arm.



**Figure 19: Temperatures of manikin body parts**

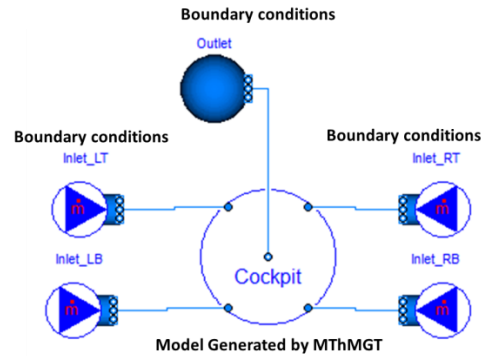
### 4.2 Cockpit Ventilation

In order to determine the required ventilation performance of the aircraft air conditioning system, heat loads have to be balanced with the airflow requirements in the most challenging operating conditions. Major heat sources in the aircraft cabin are passengers, electrical devices, solar radiation, indirect heating from piping and additional driving factors such as hot and humid environmental conditions on ground or at low flight level. Lightings in the cabin and IFE are also major heat sources. The MThMGT allows investigating aircraft cooling strategies.



**Figure 20: CAD and Zonal Grid of an aircraft cockpit**

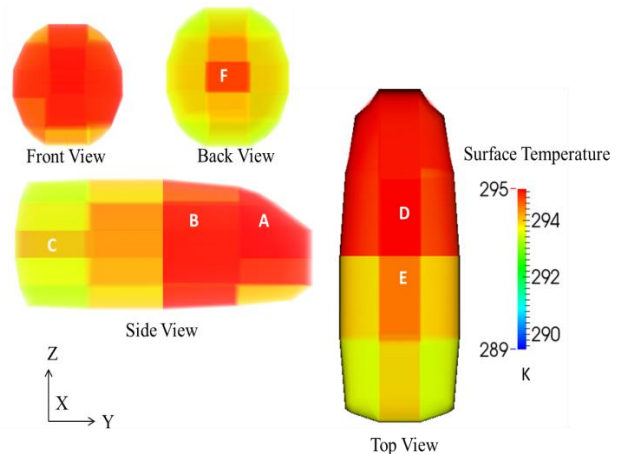
Figure 20 shows the CAD geometry of an aircraft cockpit (top). The cockpit is subdivided into 4x3x5 zones assuring that each EQS is located in a separate zone (bottom). Heat productions of EQS1, EQS2 and EQS3 are set to 2.0, 1.0 and 1.5 kW respectively.



**Figure 21: Auto-Generated Thermal Model**

Figure 21 shows the generated thermal model connected to the airflow sources and the pressure sink. There are four airflow inlets in the cockpit: two at the top and two at the bottom. The outlet is located in the center of the rear side. The user needs to define boundary conditions such as inlet airflow rate, inlet air temperature, sink air pressure and exterior temperature and to select an appropriate wall model from a wall model package.

Simulation results are exported into the Paraview visualization format for post-processing [8]. Figure 22 shows simulated wall inner surface temperatures of the cockpit. The impact of radiation from equipment simulators is reflected in results. Facets A, B and D are in close vicinity to EQS1 and EQS2. Facet C and F are influenced by EQS3. Facet E receives radiation from both EQS2 and EQS3.



**Figure 22: Wall inner surface temperatures**

Figure 23 shows simulated air node temperatures in the cockpit. EQS1, EQS2 and EQS3 are located in zones A, B and C respectively. In these zones, air temperature is predicted to be higher than in the other, empty zones.

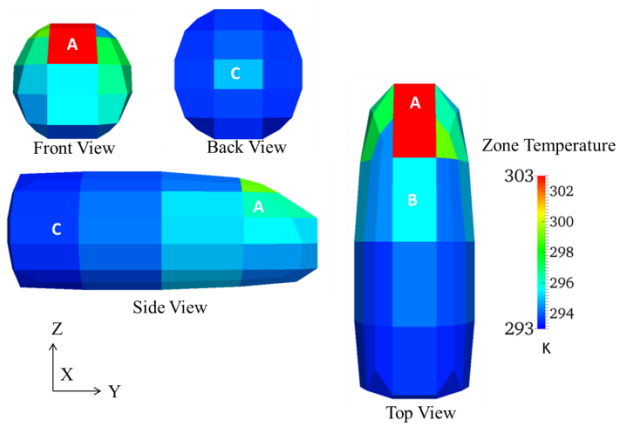


Figure 23: Air node temperatures

## 5 Conclusion and Outlook

A tool to model airflow and radiative heat transfer for complex geometries in 3D space has been outlined in the presented work. The Modelica Thermal Model Generation Tool closes the gap between the CAD model and the resulting Modelica thermal model. The exported model shows to correctly predict the interaction of radiative and convective heat transfer in an experimental validation case. The possibility to model the impact of radiative heating and to assess the impact of different airflow pattern on heating and cooling has been proved for both rectangular and irregular geometries.

The disadvantage of the generated code is that it is plain text with only scarce use of the graphical programming possibilities of Modelica. Further customization or changes in the model therefore require a high level of expertise of the user both in terms of the logics of the generated model and the code of contained models.

To keep the model accessible even for less experienced users, an export script allowing the visualization of simulation results in the open source post processing software ParaView has been written. From this visualization, the user can for example assess cooling and heating strategies or find local hot or cold spots.

For the future, we intend to include a model which can calculate convective heat transfer coefficients considering local zonal airflow pattern, surface properties and its orientation. This development provides a tool to predict thermal comfort in enclosed 3D spaces. This tool will allow for better thermal modelling when considering thermal management in buildings, automobiles or aircrafts.

## 6 Acknowledgements

This research is benefiting from the work that is being done in the European Community's Clean Sky JTI under grant agreement n° CSJU-GAM-ED-2008-001.

## 7 References

- [1] Norrefeldt, V., Andersson, D., Pathak, A. et al.: The Indoor Climate Library and its application to heat and moisture transfer in a vehicle cabin, 9<sup>th</sup> Modelica Conference, Munich, Germany, 03.-05. september 2012
- [2] Wetter, M.: Modelica library for building heating, ventilation and air-conditioning systems, 7<sup>th</sup> International Modelica Conference, Como, Italy, 20.-22. september 2009
- [3] Wetter, M.: Multizone airflow model in Modelica, 5<sup>th</sup> Modelica Conference, Vienna, 4.-5.september 2006
- [4] Bonvini, M., Leva, A.: Object-oriented sub-zonal room models for energy-related building simulation, 8<sup>th</sup> Modelica Conference, 20.-22. march 2011, Dresden, Germany
- [5] Wetter, M., Zuo, W., Nouidui, T.: Modeling of heat transfer in rooms in the Modelica "Buildings" Library, Building Simulation, Sydney, Australia, 14.-16. november 2011
- [6] Norrefeldt, V., Grün, G.: VEPZO - Velocity Propagating Zonal Model for the prediction of airflow pattern and temperature distribution in enclosed spaces, 9<sup>th</sup> Modelica Conference, Munich, Germany, 03.-05. september 2012
- [7] Pathak, A., Norrefeldt, V., Grün, G.: Modelling of radiative heat transfer in Modelica with a mobile solar radiation model and a view factor model, 9<sup>th</sup> Modelica Conference, September 3-5, Munich, Germany
- [8] Paraview: Paraview, [www.paraview.org](http://www.paraview.org), page consulted 05.12.2013,

# Modelling long-wave radiation heat exchange for thermal network building simulations at urban scale using Modelica

Moritz Lauster, Peter Remmen, Marcus Fuchs, Jens Teichmann, Rita Streblov, Dirk Müller  
RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings  
and Indoor Climate  
Mathieustr. 10, 52074 Aachen, Germany  
mlauster@eonerc.rwth-aachen.de

## Abstract

There are different options for modelling indoor and outdoor long-wave radiation exchange in thermal building models for simulations at urban scale. For improving these building models, a good trade-off between accuracy and simulation time is a major challenge. To evaluate different radiation models for thermal network building models, we compared four outdoor radiation and two indoor radiation models.

For the comparison, we set-up three test cases on a generic room and a single family dwelling and analysed surface temperatures, heat demands, and simulation times. The results favoured an outdoor radiation exchange model according to the German Guideline VDI 6007 with modified parameter calculations. It includes important simplifications that lead to short computing time while keeping a sufficient accuracy. For indoor radiation exchange modelling at constant temperatures, a linear approach significantly reduces simulation time without any major accuracy losses.

*Keywords: thermal network building model, equivalent outdoor temperature, long-wave radiation exchange, building performance simulation*

## 1 Introduction

One current challenge in the field of building simulation is the thermal simulation of entire city districts. For this task, simplified building models are an interesting approach as they require comparably low parameterization and computational efforts. A comprehensive discussion of this topic can be found in [1, 2], which summarize the state-of-the-art in dynamic building simulation. The approaches to simplify building models include thermal network models, which are based on analogies to electrical problems and have successfully been applied at urban scale [3, 4].

Thermal simulation at urban scale aims at understanding and efficiently directing energy flows between different subsystems like generation units and buildings. Of particular interest are holistic control strategies and heat storage effects, because they offer potential for improving the energy system without the need for high investment. To investigate such measures, dynamic simulations at variable time step seem more promising than static and quasi-static calculations. Nevertheless, traditional building simulation environments are often limited to an hourly time step. Thus, a growing community is developing model libraries to simulate building performance at building and urban scale using Modelica [5, 6, 7, 8], often using thermal network models to describe thermal building behaviour.

Within common thermal network models, the long-wave radiation heat fluxes on wall surfaces can be considered on different levels of detail. These heat fluxes are the result of temperature differences caused by indoor and outdoor sources such as heating systems and solar radiation. Different approaches exist for the calculation of radiative heat exchange. Some models use Stefan-Boltzmann law while others use linearized radiation equations. Thus, these models differ in the accuracy of their results as well as in computation time. So far, it has not been determined which of these models is best suited for thermal simulation at urban scale.

In this paper, we present four approaches to consider long-wave radiation exchange in building simulation. The aim is to evaluate these approaches regarding their suitability for thermal network building models and urban-scale applications by means of a balanced trade-off between physical resolution and calculation time [9].

Firstly, we present the methodology and modelling assumptions. Afterwards, we implement four models and define a benchmark test case. Finally, we discuss the simulation results and end with conclusions.

## 2 Modelling approaches

### 2.1 Building model validation

Thermal network models describe heat transfer and storage problems with a number of thermal resistances and capacitances. While highly discretized models provide high spatial resolution, low order models require less computational effort at the cost of accuracy.

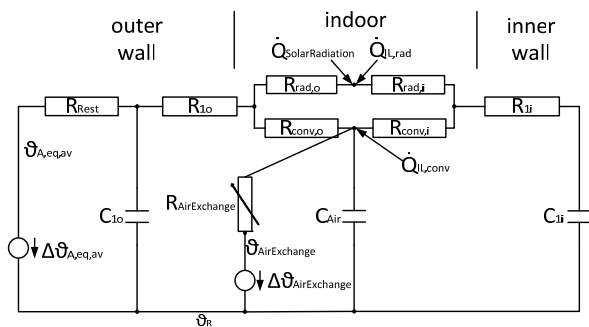


Figure 1: Low order building model derived from VDI 6007

In this study, we use a building model based on the German Guideline VDI 6007 [10] as seen in Figure 1. The model divides the building mass into two capacitances representing all internal and external building elements respectively. The heat transfer through the outer wall is described by two resistances while another resistance is used to damp the adiabatic inner wall capacity. The indoor heat exchange between the walls and the air node can be calculated in different ways. While the VDI 6007 defines a combined coefficient of heat transfer, we distinguish between radiative and convective heat transfer (Figure 1). Outdoor radiation sources like solar radiation are considered via an adapted equivalent outdoor air temperature  $\vartheta_{A,eq,av}$  (in the following referred to as  $T_{eq}$ ). Substituting the outdoor air temperature with this equivalent outdoor air temperature is a way to incorporate the effects of long-wave radiation into the model.

While we kept most parts of the theory and model description given in VDI 6007, we did not follow the given analytical equations [11]. We rather took advantage of Modelica's abilities to formulate acausal equations in an object-oriented structure. We defined a sub-model for each element in Figure 1 and connected them to the circuit shown in Figure 6. Each sub-model describes either heat transfer phenomena (resistance) or storage effects (capacity). In order to validate this thermal network model, we performed benchmark tests according to the American Standard ASHRAE 140 [12]. This standard provides a set of test cases and

corresponding results of standard building models. If deviations in the validation process exceed given limits, the standard suggests further test procedures for each test case. In this way, the standard supports the identification of sub-models with optimisation potential. The tests gave valid results for most test cases. Nevertheless, the validation process identified problems with the handling of radiation exchange. One way to address this issue would be more detailed radiation exchange models, while potential simplifications should still be considered. Otherwise, radiation and building model could be out of balance by means of level of detail and required computational costs.

We identified long-wave radiation exchange as one key part for optimization. It seems to have major influence on heat demand [13] and different modelling approaches are available. As the ASHRAE provides no test case solely for the effects of long-wave radiation heat exchange, we refined one ASHRAE in-depth test set-up to focus only on long-wave radiation.

In the following sections, we will discuss and compare different approaches for outdoor as well as indoor long-wave radiation heat exchange.

### 2.2 Outdoor long-wave radiation exchange

According to VDI 6007, the heat flux due to ambient radiation sources on the outer walls can be considered in an equivalent adapted outdoor temperature.

Figure 2 shows typical influences that need to be considered in the adapted outdoor temperature. Radiation can be divided into short-wave and long-wave. In contrast to short-wave radiation, measurement data of long-wave radiation sources are rarely available. Furthermore, long-wave radiation has to be subdivided into atmospheric, ambient and partly reflected ground radiation. Thus, empirical methods are used to consider the long-wave heat flow as a function of outdoor temperature and cloud coverage [14, 15, 2].

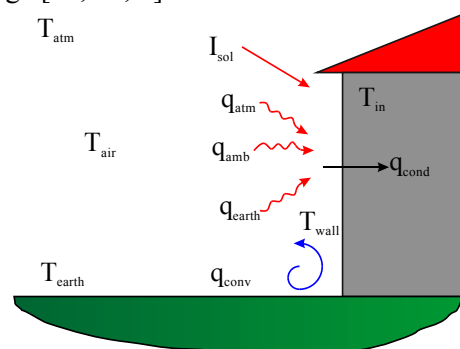


Figure 2: Heat flows on the outer wall

According to Figure 2 a heat balance of the wall consists of:

$$q_{cond} = q_{short-wave} + q_{long-wave} + q_{conv} \quad (1)$$

Starting from this heat balance it is possible to implement models with different levels of simplifications (see Figure 3).

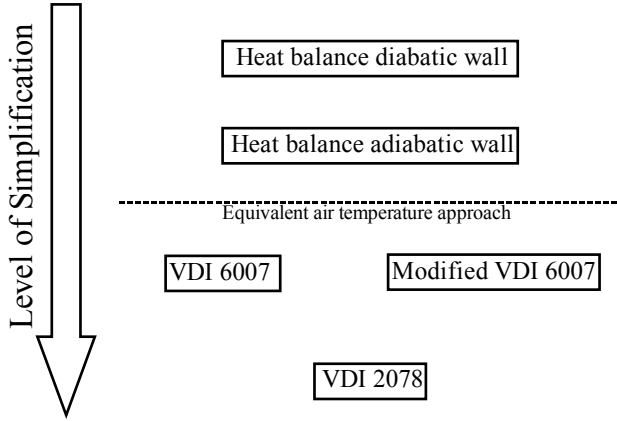


Figure 3: Schematic diagram of investigated models and their level of simplification

As a first simplification the outside wall can be considered as adiabatic, which leads to  $q_{cond} = 0$ . In the following subsections we describe four ways to model long-wave radiation heat exchange. The meaning of the variables can be found in the nomenclature.

### 2.2.1 Heat Balance

The three occurring heat flows can be written as:

$$q_{conv} = \alpha_{conv} \cdot (T_{air} - T_{wall}) \quad (2a)$$

$$q_{long-wave} = \sigma \cdot \varepsilon_{wall} \cdot (\varphi_{atm} \cdot (T_{atm}^4 - T_{wall}^4) + \varphi_{earth} \cdot (T_{earth}^4 - T_{wall}^4) + \varphi_{amb} \cdot (T_{air}^4 - T_{wall}^4)) \quad (2b)$$

$$q_{short-wave} = a_{abs} \cdot I_{sol} \quad (2c)$$

$\varphi_i$  is a view factor between the wall and considered long-wave radiation source. The temperature of extra-terrestrial sources, also called sky temperature, can be calculated as a function of long-wave radiation heat flux. Furthermore, the earth temperature can be regarded as equivalent to the outdoor air temperature [16, 17]:

$$T_{atm} = \left( \frac{E_{atm}}{\sigma} \right)^{0,25} \quad (3a)$$

$$T_{earth} = T_{air} \quad (3b)$$

Inserting Equations 2 a-c and 3a/b in Equation 1 provides the heat balance of the adiabatic outside wall. This heat balance can be iteratively solved for  $T_{wall}$ . Without any further simplifications, this Heat Balance method provides a relatively detailed model.

### 2.2.2 Equivalent air temperature according to VDI 6007

A widely used simplification is to describe the occurring heat flows with one equivalent heat flux [10, 18, 19]. Introducing a combined radiative and convective coefficient of heat transfer and an equivalent temperature leads to:

$$\alpha_{comb} \cdot (T_{eq} - T_{wall}) = q_{short-wave} + q_{long-wave} + q_{conv} \quad (4)$$

Expressing the equivalent temperature according to VDI 6007 requires further assumptions. The long-wave radiation heat exchange is linearized using  $T_{atm}$  and  $T_{earth}$ , the wall temperature is set equal to the outdoor air temperature and  $T_{atm}$  and  $T_{earth}$  are calculated from long-wave radiation heat flows.

$$T_{atm} = \left( \frac{E_{atm}}{\varepsilon_{earth} \cdot \sigma} \right)^{0,25} \quad (5a)$$

$$T_{earth} = \left( \frac{E_{earth}}{\varepsilon_{earth} \cdot \sigma} \right)^{0,25} \quad (5b)$$

$$\alpha_{rad} = \frac{T_{atm}^4 - T_{earth}^4}{(T_{atm} - T_{earth})} \cdot \sigma \cdot \varepsilon_{wall} \quad (5c)$$

Transforming the equation (4) to  $T_{eq}$  and apply the listed assumptions, the equivalent temperature can be solved as follows:

$$T_{eq} = T_{air} + \frac{\alpha_{rad}}{\alpha_{comb}} \cdot \quad (6)$$

$$\left( (T_{atm} - T_{air}) \cdot \varphi_{atm} + (T_{earth} - T_{air}) \cdot \varphi_{earth} \right) + \frac{q_{short-wave}}{\alpha_{comb}}$$

### 2.2.3 Modified equivalent air temperature based on VDI 6007

Equation 6 implies a constant combined coefficient of heat transfer. As a further improvement,  $\alpha_{comb}$  is computed as the sum of constant  $\alpha_{conv}$  and a variable coefficient for long-wave radiation heat transfer  $\alpha_{rad}$ .

In addition, both  $T_{atm}$  and  $T_{earth}$  are computed by dividing them through the emissivity factor of the earth. According to [16, 17], it is more accurate to calculate the sky temperature with Equation 3a. These changes from the originally VDI 6007-model are summarized under the variant name Mod VDI 6007.

### 2.2.4 Combined outdoor temperature according to VDI 2078

The revised German Guideline VDI 2078 describes cooling load calculations and provides a combined outdoor temperature. Formally, this temperature originates from a similar approach as the VDI 6007. However, the VDI 2078-model is further simplified by neglecting any changes in sky emissivity or in the coefficient of long-wave radiation heat exchange. Sky emissivity is set to 0.74, which holds roughly

true for summer [20]. Besides, the air temperature is taken as the direct reference temperature for all calculated heat fluxes. As this model represents a further simplification, we include it in our study and compare its results to the other three models.

$$T_{eq} = T_{air} - \frac{\sigma \cdot \epsilon_{wall}}{\alpha_{comb}} \cdot ((T_{air,m})^4 + (T_{air} - T_{air,m}) \cdot 1.05) \cdot (1 - \varphi_{atm} \cdot \epsilon_{atm} - \varphi_{earth} \cdot \epsilon_{earth}) + \frac{q_{short-wave}}{\alpha_{comb}} \quad (7)$$

### 2.3 Indoor long-wave radiation exchange

The long-wave radiation heat exchange within the building is commonly modelled in two different ways. One possibility, described in the VDI 6007 guideline, suggests calculating this way of heat exchange with a constant coefficient of heat transfer between inner and outer wall. The guideline prescribes a value of 5 W/(m<sup>2</sup> K), which can also be found in [21]. It holds true for non-metallic materials with an emissivity factor of 0.8 and a mean temperature of 26.85°C. This method does not take into account the temperature dependency of long-wave radiation heat exchange as formulated in the Stefan-Boltzmann law.

Alternatively, a heat balance between inner wall, outer wall, indoor radiation sources and transmitted solar radiation can be formed.

## 3 Implementation

For this study, we use the Modelica Standard Library and self-developed building model libraries. An overview of these libraries is given in [6, 22].

As all radiation heat exchange models described in Section 2 derive from a similar theoretical approach, we are able to define a partial model as a general base class (Figure 4).

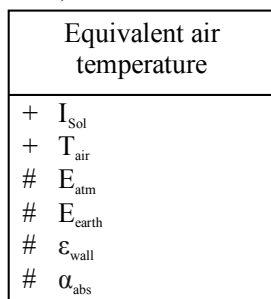


Figure 4: UML diagram of equivalent air temperature base class

The four different long-wave radiation exchange models all extend from this base class. This gives us the advantage to define a replaceable block for long-wave radiation heat transfer (1) connected to our building physics model (2) which can then be set to

use one of the four radiation models (Figure 5). The building model is connected via heat ports to the long-wave radiation heat transfer (a). Both are connected with a weather model, which provides solar radiation (b), outdoor air temperature, sky and terrestrial long-wave radiation (c). Furthermore, the building model needs additional information about infiltration rates (d), as well as about convective (e) and radiative (f) inner loads via heat ports.

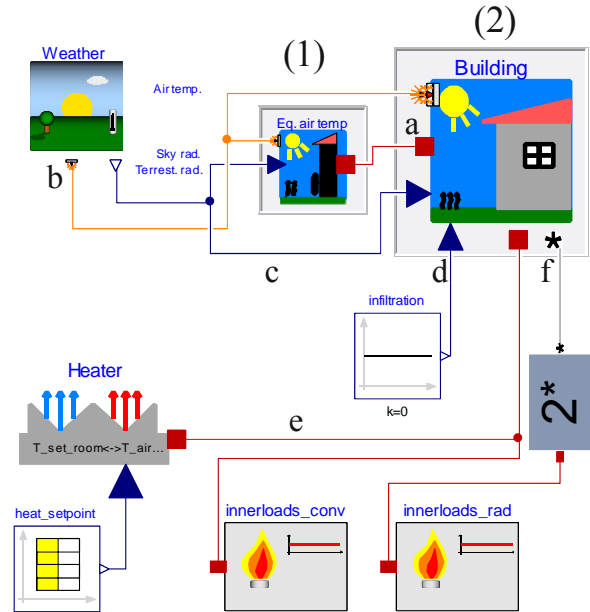


Figure 5: Implementation in Dymola

Taking advantage of separated definitions of convective and radiative heat transfer (see Section 1), we can easily change the definition of long-wave radiation heat exchange (see Figure 6).

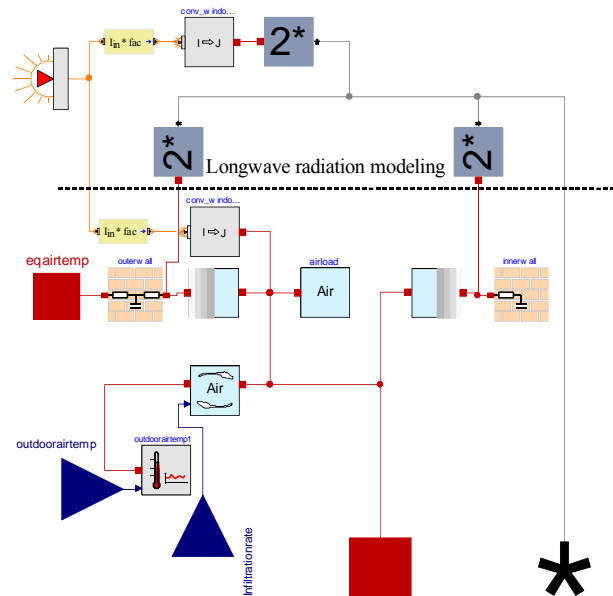


Figure 6: Zone model in Dymola

In the context of city district simulations, the object-oriented approach of Modelica is an important



advantage. It allows a fast and easy use of the same model in several instances while adapting the sub-models to the requirements of each instance.

A further advantage besides variable time step and object-orientation supported by Modelica in this context is acausal modelling. Energy supply systems at urban scale are relatively complex and flow directions for energy and mass flows are usually unknown.

## 4 Test case and results

### 4.1 Test case

To figure out the differences between the models and identify the most promising approach for our building model, we developed three test cases. Test Case 1 and Test Case 2 are based on the “Case 220: In-Depth Series Base Case” from ASHRAE 140. As most ASHRAE 140 in-depth tests, it is based on a single room consisting of five light-weight outer walls only. We simulated 1<sup>st</sup> July of the typical meteorological year (TMY) provided with the standard [12]. Before and after the simulated day all initial temperatures are held constant. For Test Case 1 the outside wall is only subject to outdoor air temperature and long-wave radiation. Convection and absorption of short-wave radiation are not considered. Thus Test Case 1 uses outdoor air temperature and long-wave radiation from the atmosphere and the earth as input-data. Test Case 2 differs in the number of heat transfer mechanisms which affect the building envelope. Convection and absorption of short-wave radiation are additionally taken into account. Further inputs are thus solar radiation on tilted surfaces and combined as well as convective coefficients of heat transfer (set to 28.5 W/(m<sup>2</sup> K) and 24.67 W/(m<sup>2</sup> K) for walls, and 16.37 W/(m<sup>2</sup> K) for windows respectively). Only one day is simulated as well, all temperatures are held constant before and after that day. Other parameters of Test Cases 1 and 2 are listed in Table 1.

Table 1: Boundary conditions of Test Case 1 and 2

PARAMETER	VALUE
Infiltration	0 m <sup>3</sup> /h = const.
Internal gains	0 W = const.
Temperature of floor coupling	T <sub>air</sub>
$\alpha_{\text{abs}}$	0.6
$\epsilon_{\text{wall}}$	0.90
mech. equipment	none

The purpose of Test Case 3 is to emphasize on the inside long-wave radiation heat exchange, heat

demand and simulation time. Since the simplified building model represents all outer and inner walls by one capacitance per class (see Section 2.1), radiation exchange can only occur if elements of both classes exist. We thus set up a test case with outer as well as inner walls. Test Case 3 is based on a two storey single-family dwelling with a living area of 150 m<sup>2</sup>. The building has a high thermal mass and is well insulated according to German Energy Savings Ordinance 2009. A full year is simulated. Calculation of the equivalent air temperature is performed with convective and long-wave radiation heat transfer as well as short-wave absorption. Combined and convective coefficients of heat transfer are set to 25 W/(m<sup>2</sup> K) and 20 W/(m<sup>2</sup> K) respectively. The boundary conditions for this case can be found in Table 2.

Table 2: Boundary conditions of Test Case 3

PARAMETER	VALUE
Infiltration	0 m <sup>3</sup> /h = const.
Internal gains	0 W = const.
Temperature of floor coupling	10.36°C = const.
$\alpha_{\text{abs}}$	0.38
$\epsilon_{\text{wall}}$	0.90
mech. equipment	ideal heater
Thermostat strategy	22°C (6h – 20h) 17°C (20h – 6h)

All simulations are performed on a computer with following technical data (see Table 3).

Table 3: Data of used equipment

CHARACTERISTIC	VALUE
Operating system	Windows 7
Number of processors	4
Clock speed	2.67 GHz
Working memory	4 GB

To quantify differences between the models, a root-mean-square deviation (RMSD) is used. When comparing more than two models it is appropriate to form the RMSD between the minimum and the maximum of each hour. In this way a range within which the temperatures lie is calculated (RMSD-R).

### 4.2 Limitations

The major limitations in the presented work are:

- Subject of this analysis are only single-zone buildings with specific characteristics
- All models are based on assumptions; the results are compared between each other but not to measurements.

- Empirical equations are used to calculate long-wave radiation as a function of outdoor air temperature. Thus, all models are dependent either directly or indirectly on outdoor air temperature.
- Heat balances for the models relate to slightly different locations (wall surface or nearby the surface) and cannot be harmonized.
- Convective heat transfer  $\alpha_{\text{conv}}$  is regarded constant. In reality, it may vary over time depending on wind speed and direction.
- Our presented analysis is limited to single days and overall yearly values.

### 4.3 Outdoor long-wave radiation exchange

To understand the differences between the models, we investigated the surface temperatures on the building envelope as direct model outputs. The results of Test Case 1 with long-wave radiation load can be obtained in Figure 7. While the temperature of the VDI 6007-model holds the highest values, the model of guideline VDI 2078 holds the lowest. The RMSD between these two models is 11.24 K. The high temperatures of the VDI 6007-model close to the outdoor air temperature are striking. In other investigations a rise above the outdoor air temperature was detected, which contradicts observation [23]. The Mod VDI 6007-model has a clearly lower course and converges to the most detailed Heat Balance-model. It is apparent that Mod VDI 6007 differs the most at high temperatures from Heat Balance. The RMSD between these two models is 0.54 K in contrast to 2.56 K for VDI 6007 and Heat Balance.

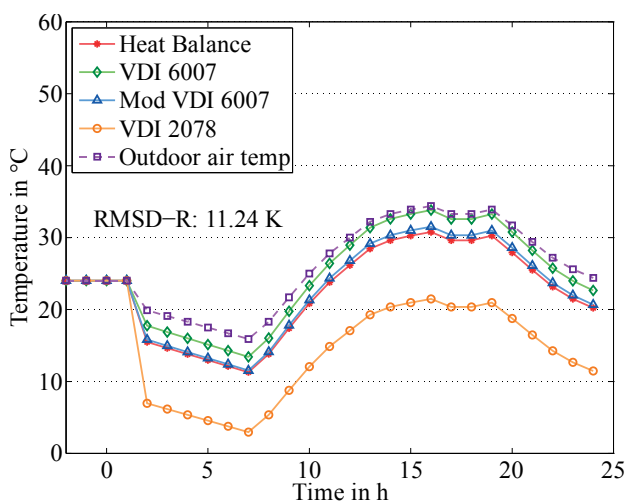


Figure 7: Equivalent outdoor temperature on July,1. of TMY; long-wave load

The comparably low temperatures of VDI 2078 can be explained by the handling of sky emissivity. The

emissivity is assumed to be constant, which cannot be guaranteed for the investigated day. Sky conditions or reflection of long-wave radiation are not taken into account. Furthermore, the long-wave radiation heat transfer coefficient is not calculated dependent on ambient conditions.

Modifications in Mod VDI 6007 compared to VDI 6007 primarily concern long-wave radiation exchange. They cause a shift in the results towards the Heat Balance-model. Both models, Mod VDI 6007 and Heat Balance, calculate the sky temperature in the same way. Small differences at high temperatures can be justified as the Mod VDI 6007-model does not take a temperature depending emission of the wall into account.

Test Case 2 is a more realistic test case as it takes convection and short-wave radiation into account (see Figure 8). The aim is to analyse the impact of long-wave radiation under real conditions. As expected, the differences between the models decrease due to the new influences. Mainly responsible for this is convective heat transfer. This heat flux is calculated with a constant coefficient of 24.67 W/(m<sup>2</sup> K) for walls and 16.37 W/(m<sup>2</sup> K) for windows in all models, hence it is about 3-6 times higher than radiative heat transfer. However, the VDI 6007 and the VDI 2078 still represent the extremes. The RMSD between these two models decreases to 2.22 K. Especially the extreme rise of VDI 2078 in contrast to the first test case is striking. The difference in the temperature after 24 h amounts to over 10 K. On the one hand this shows the overestimation of long-wave radiation heat transfer of VDI 2078. On the other hand it displays the effect of convective heat transfer. Comparing Mod VDI 6007 and the Heat Balance, a difference is hardly recognizable (RMSD: 0.1 K). All adapted temperatures rise over the ambient air temperature during the day, resulting from the short-wave radiation absorption.

Regarding the given test cases, the results justify the implemented simplifications in the VDI 6007 and Mod VDI 6007-models. Only the VDI 2078 shows major differences to the most complex Heat Balance-model.

As expected, the convective heat transfer smoothes the equivalent temperatures of all models in Test Case 2.

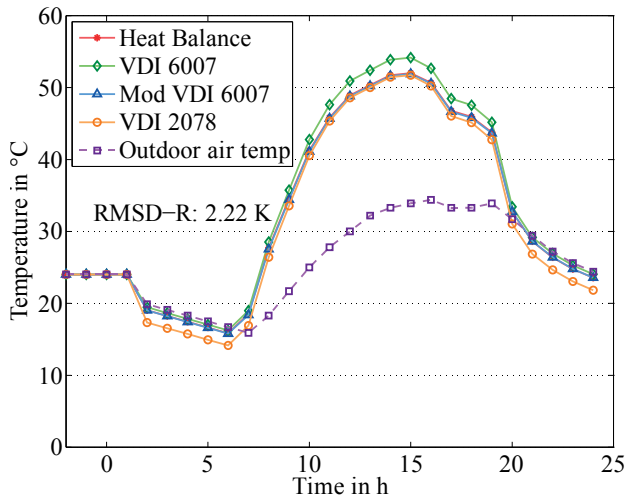


Figure 8: Equivalent outdoor temperature on July 1 of TMY; long-wave, short-wave and convective load

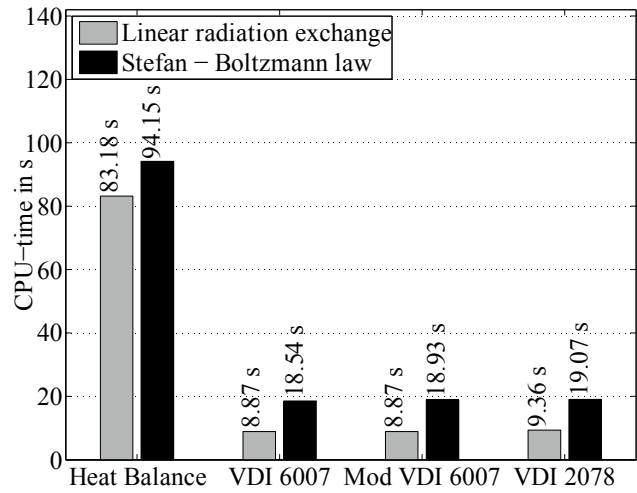


Figure 9: Comparison of linear radiation model and Stefan-Boltzmann law for indoor radiation exchange

An overestimation of long-wave radiation heat exchange of VDI 2078 can be clearly observed. The rise over air temperature during the day in all models is caused by short-wave radiation absorption of the wall. In both test cases the strong dependency on the air temperature can be monitored.

In conclusion the Mod VDI 6007 model keeps the best accuracy compared to the complex Heat Balance-model while including important simplifications that lead to shorter computing times (see Figure 9).

#### 4.4 Indoor long-wave radiation exchange

Besides the outdoor long-wave radiation heat exchange, different options exist to model indoor radiation exchange. In our analysis, we focus on heat demand and simulation time.

Figure 9 shows the CPU-time for a one-year simulation of Test Case 3. We simulated each equivalent temperature model with both explicit heat balance following the Stefan-Boltzmann law and a linearized approach. The CPU-time decreases using the linearized approach for indoor radiation. The percentages of savings in simulation time for simplified outdoor models are particularly remarkable. For the VDI 6007-model the simulation time is 52.16% shorter compared to the detailed indoor heat balance. The differences in simulation time result from the removal of the heat balances calculated with the Stefan-Boltzmann law. These heat balances contain a dependency on  $T^4$ , which is solved iteratively. This is also the reason for the massive increase of needed time using Heat Balance-model for the outdoor heat exchange.

As we control the indoor temperatures on a level between 17°C and 22°C, the chosen coefficient of radiative heat transfer is justified (see Section 2.3). Hence, the different modelling approaches lead to no major differences in the simulated heat demand. Comparing the indoor models to each other the deviation over one year is below 2.1%. This suggests using a linear approach for indoor radiation when keeping the indoor temperature on a controlled level is a valid simplification. Nevertheless, it is important to choose a corresponding coefficient of radiative heat transfer.

## 5 Conclusion

There are different options for modelling the indoor and outdoor long-wave radiation exchange in thermal building models for simulations at urban-scale. For improving these building models, a good trade-off between accuracy and simulation time is a major challenge. The main differences of common radiation modelling assumptions concern the impact of outdoor radiation sources and linearizing the Stefan-Boltzmann law.

To evaluate different radiation models for thermal network building models, we compared four adapted outdoor air temperature models. In addition, we investigated the handling of indoor radiation exchange with the Stefan-Boltzmann law and a linear approach.

For the comparison, we set-up three test cases on a generic room and a single family dwelling and analysed surface temperatures, heat demands and simulation times. We varied the number of radiation sources between the test cases to observe radiation heat exchange under generic and real conditions.

Modelica proved to be a promising modelling language for urban-scale building simulations. We identified three major prerequisites and advantages:

- Allowing solvers with variable time steps
- Use of object-oriented modelling approaches
- Use of acausal modelling approaches

The results of the test cases show promising potential for an outdoor radiation exchange model based on a modified approach from German guideline VDI 6007. It includes important simplifications that lead to short computing time while keeping a sufficient accuracy. For indoor radiation exchange modelling at constant temperatures, a linear approach significantly reduces simulation time without major losses in accuracy. An additional comparison with measurement data could further help to validate this suggested approach.

## Acknowledgements

We gratefully acknowledge the financial support for this project by BMWi (German Federal Ministry of Economics and Technology) under promotional reference 03ET1004A.

## Nomenclature

$a_{\text{abs}}$	coefficient of short-wave absorption	$W/(m^2 K)$
$\alpha$	coefficient of heat transfer	$W/(m^2 K)$
$E$	long-wave radiation	$W/m^2$
$\varepsilon$	emissivity factor	-
$I_{\text{sol}}$	solar radiation on tilted surface	$W/m^2$
$\sigma$	Stefan-Boltzmann-Factor	$W/(m^2 K^4)$
$T$	temperature	K
$q$	heat flux of wall	$W/m^2$
$\varphi$	view factor of long-wave radiation source	-
air	outdoor air	
amb	ambient source	
atm	extraterrestrial source	
cond	conduction	
conv	convection	
comb	combined	

earth	terrestrial source
eq	equivalent
m	daily average
rad	long-wave radiation

## References

- [1] Hensen J, Lamberts R. Building performance simulation for design and operation. Abingdon, Oxon, New York, NY: Spon Press; 2011.
- [2] Clarke JA. Energy simulation in building design. 2nd ed. Oxford: Butterworth-Heinemann; 2001.
- [3] Robinson D (ed.). Computer modelling for sustainable urban design: Physical principles, methods and applications. 1st ed. London: Earthscan; 2011.
- [4] Kämpf JH, Robinson D. A simplified thermal model to support analysis of urban resource flows. *Energy and Buildings* 2007;39(4):445–53.
- [5] Wetter M, van Treek C. IEA EBC Annex 60. [November 15, 2013]; Available from: <http://www.iea-annex60.org/>.
- [6] Müller D, Hosseini Badakhshani A. Gekoppelte Gebäude- und Anlagensimulation. In: Proceedings BauSim Conference; 2010.
- [7] Nytsch-Geusen C, Huber J, Ljubijankic M, Rädler J. Modelica Buildingsystems - eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. In: Proceedings BauSim Conference 2012; 2012, p. 271–278.
- [8] Wetter M, Zuo W, Nouidui TS. Recent Developments of the Modelica "Buildings" Library for Building Energy and Control Systems. In: Proceedings 8th Modelica Conference; 2011, p. 266–275.
- [9] Bonvini M, Leva A. Exploiting Object-Oriented Modelling for Scalable-Detail Studies on Control for Energy Efficiency. In: Proceedings 2012 IEEE Multi-conference on Systems and Control; 2012, p. 770–775.
- [10] German Association of Engineers. Calculation of transient thermal response of rooms and buildings - Modelling of rooms;91.140.10(VDI 6007-1). Düsseldorf: Beuth Verlag GmbH; 2012.
- [11] Lauster M, Teichmann J, Fuchs M, Streblov R, Mueller D. Low order thermal network models for dynamic simulations of buildings on city district scale. *Building and Environment* 2014;73:223–31.

- [12] American Society of Heating, Refrigerating and Air-Conditioning Engineers. Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs(ASHRAE 140-2007). Atlanta: ANSI/ASHRAE; 2007.
- [13] Shi Z, Zhang X. Analyzing the effect of the longwave emissivity and solar reflectance of building envelopes on energy-saving in buildings in various climates. *Solar Energy* 2011;85(1):28–37.
- [14] German Association of Engineers. Environmental Meteorology;13.040.10(VDI 3789-2). Düsseldorf: Beuth Verlag GmbH; 1994.
- [15] Christoffer T, Deutschländer M., Webs M. Testreferenzjahre von Deutschland für mittlere und extreme Witterungsverhältnisse TRY. Deutscher Wetterdienst, Offenbach 2004.
- [16] EnergyPlus Development Team. EnergyPlus Engineering Documentation. [November 15, 2013].
- [17] Müller E. Development of a test reference year on a limited data base for simulations on passive heating and cooling in Chile. In: *Proceedings Building Simulation Conference*; 2001.
- [18] Nehring G. Über den Wärmefluß durch Außenwände und Dächer in klimatisierten Räumen infolge der periodischen Tagesgänge der bestimmenden meteorologischen Elemente. *Gesundheits Ingenieur* 1962;83(7):185–216.
- [19] Mackey CO, Wright LT, Ithaca NY. Periodic heat flow - homogeneous walls or roofs. *Transactions American Society of Heating and Ventilating Engineers* 1944(50):293–312.
- [20] German Association of Engineers. Cooling Load Calculation of Air-conditioned Rooms;91.140.30(VDI 2078-1). Düsseldorf: Beuth Verlag GmbH; 1996.
- [21] German Association of Engineers. VDI heat atlas. 2nd ed. Heidelberg: Springer; 2010.
- [22] Lauster M, Streblov R, Müller D. Modelica-Bibliothek und Gebäudemodelle. In: *Proceedings Symposium Integrale Planung und Simulation in Bauphysik und Gebäudetechnik*: 26.03. - 28.03.2012, Technische Universität Dresden; 2012.
- [23] Norbert Nadler. Validierung des Rechenkerns der C.A.T.S.-Kühllastberechnung anhand der neuen VDI 6007-1. *HLH Lüftung/Klima - Heizung/Sanitär - Gebäudetechnik* 2013;64(1):36–41.



# Extension of the FundamentalWave Library towards Multi Phase Electric Machine Models

Christian Kral  
Electric Machines, Drives and Systems  
1060 Vienna, Austria  
dr.christian.kral@gmail.com

Anton Haumer  
Technical Consulting  
3423 St.Andrä-Wördern, Austria  
a.haumer@haumer.at

Reinhard Wöhrnschimmel  
AIT GmbH  
1210 Vienna, Austria

## Abstract

Electric machine theory and electric machine simulations models are often limited to three phases. Up to the Modelica Standard Library (MSL) version 3.2 the provided machine models were limited to three phases. Particularly for large industrial drives and for redundancy reasons in electric vehicles and aircrafts multi phase electric machines are demanded. In the MSL 3.2.1 an extension of the existing FundamentalWave library has been performed to cope with phase numbers greater than or equal to three. The developed machine models are fully incorporating the multi phase electric, magnetic, rotational and thermal domain. In this publication the theoretical background of the machines models, Modelica implementation details, the parametrization of the models and simulation examples are presented.

*Keywords: Modelica Standard Library, multi phase, electric machine models, induction machine, synchronous machine, synchronous reluctance machine*

## 1 Introduction

Three phase induction, synchronous and synchronous reluctance machines are state of the art solutions for industrial applications, traction drives in electric vehicles, railways, trams, and underground trains, as well as air craft motors and generators. If a full leg of the supplying three phase converter fails, the machine cannot be operated any more, once stopped. In particular, the higher ambient temperatures of traction machines may cause the power electronics to fail. In order to overcome machine outage due to a single converter leg failure, phase numbers greater than three may be used for electric machines and power electronics. In the following multi phase will indicate phase numbers greater than three. If it is referred to only three phases, this will be indicated explicitly.

Multi phase drives consist of the multi phase machine including an inverter with power electronics plus control. A phase number greater than three thus requires a higher number of power electronic switches, such as IGBTs or MOSFETs, etc. The higher phase numbers, however, increase

cost and add complexity to the drive structure.

However, phase numbers equal to  $2^n$  with integer  $n$  are excluded from the actual implementation. The reason for excluding these phase numbers is that for example four or eight phase machines have to be handled differently since two phase are separated by  $\pi/2$ , not  $\pi$ . In general, two different philosophies of multi phase drives exist:

First, the number of phases is divisible by two. In industry, typically, six phase machines are used to overcome maximum power limitations of power electronics supplying high power machines in the Megawatt range [1]. In this case maximum power of power electronics is doubled by using two three phase converters supplying a six phase machine. Usually, the phase winding orientations of the two three phase are spatially shifted by  $30^\circ$  in order to additionally reduce the magnitudes of space harmonics caused by the winding magneto motive force (MMF) and to reduce the torque ripple of the machine, respectively. The implementation of a six phase winding does not significantly increase cost of the electric machine compared with a three phase machine with the same power. Solely the additional winding ends have to be conducted to the terminal box. For six phase drives, the cost of power electronics increases due the double number of power electronic switches for the additional legs. For doubling the power of industrial drives the double cost of power electronics is in line with doubling the power. However, for traction machines six or nine phase machines are used for redundancy reasons [2, 3]. In this case several state of the art three phase converters can be used. The redundancy concept is then realized with standard components which is cheaper than designing the individual legs of the inverter. Yet, several three phase inverters of smaller power rating are usually more expensive than a three phase inverter of the same total power. Due to the multiple three phase inverters installation space increases, too. Yet, state of the art control for three phase drives can be adapted with relatively low effort due to modularly using three phase converters.

Second, the number of phases is not divisible by two. In this case mostly five (or seven) phase drives are used [4–6]. The drawback over a six phase inverter is that the modularity of the power electronics design is lower and thus cost may be higher and more design space may be required.

For the sake of completeness one more redundancy concept will be discussed here, even though it is not related with multi phase electric machines. Redundant drives with three phase machines may use modified topologies which either use an additional leg or additional switches to operate the machine in case of a failure. These topologies have the capability to be reconfigured when a failure occurs [7, 8]. Depending on the actual topology even the full power rating may be provided to the electric machine.

For controlling multi phase electric drives it is advantageous to control current components which represent the fundamental wave MMF and magnetic voltage, respectively [9]. The pulse width modulation scheme for multi phase converters with phase numbers not divisible by three has to be adapted so that a symmetrical supply can be achieved. A technical paper dealing in more detail with analysis of the multi phase drive control is also submitted to the Modelica 2014 conference and will be cited properly, in case it gets accepted.

In Modelica the first three phase electric machine models have been introduced with the MSL 2.1 in 2004 [10, 11]. An alternative implementation with magnetic fundamental wave phasors was introduced in MSL 3.2 in 2010 [12]. Since then in these models copper loss, (eddy current) core loss, friction loss, stray load loss, PM loss and brush loss are taken into account. Multi phase electric machine models have already been published decades ago [13, 14]. Yet, in most computer simulations tools there are currently only three phase machine models available. However, in the MSL 3.2.1 version of the package `Modelica.Magnetic.FundamentalWave` new multi phase electric machine models are introduced. In general, arbitrary phase numbers for stator (and rotor) windings may be used – excluding phase numbers equal to  $2^n$  with integer  $n$ . This article provides the theoretical background, details about the implementation, parametrization schemes and some examples.

## 2 Fundamental Wave Theory

Multi phase electric machine theory often relies on phasor transformations of currents, voltages and magnetic fluxes [15, 16]. A typical transformation is the symmetrical components of the instantaneous values. In case of fully symmetrical supply the machine equations based on the symmetrical components of instantaneous components can be simplified extensively.

The FundamentalWave machine models only consider fundamental wave effects so there is also a complex phasor representation of the fundamental wave of the magnetic flux and magnetic potential (difference), respectively. The connector definition of the FundamentalWave library shows:

```
connector MagneticPort
  "Complex magnetic port"
  Modelica.SIunits.
    ComplexMagneticPotentialDifference V_m
  "Complex magnetic potential difference";
  flow Modelica.SIunits.
    ComplexMagneticFlux Phi
```

```
"Complex magnetic flux";
end MagneticPort;
```

Please note, that the potential and flow variable of the connector represent instantaneous quantities  $\underline{\Phi} = \Phi_{re} + j\Phi_{im}$  and  $\underline{V}_m = V_{m,re} + jV_{m,im}$ . The complex magnetic quantities represent a spatial distribution of magnetic flux and magnetic potential (difference):

$$\begin{aligned}\Phi(\varphi) &= \text{Re}[(\Phi_{re} + j\Phi_{im})e^{-j\varphi}] \\ &= \Phi_{re} \cos(\varphi) + \Phi_{im} \sin(\varphi) \\ V_m(\varphi) &= \text{Re}[(V_{m,re} + jV_{m,im})e^{-j\varphi}] \\ &= V_{m,re} \cos(\varphi) + V_{m,im} \sin(\varphi)\end{aligned}$$

The complex potential (difference)  $\underline{V}_m$  introduced in the connector definition represents the total magnetic potential (difference) of all poles. This quantity can, thus, also be seen as the complex magnetic potential difference of an equivalent two pole machine. Physical interpretations of the complex magnetic phasors are presented and discussed in [12].

Voltages and currents are instantaneous quantities, so arbitrary waveforms and operating conditions are covered. Therefore, the machines can also be supplied with asymmetric voltages or currents. It is yet assumed that only fundamental wave effects due to these asymmetries are considered. Supply voltage or current imbalance give rise to time transient magnitudes of the magnetic flux and magnetic potential (difference). Each of the spatial field distributions can be interpreted as a forward and backward traveling fundamental wave component. Those effect of the two waves is correctly taken in account by the proposed approach.

Particular supply imbalances and certain asymmetries may cause magnetic flux and magnetic potential (difference) phasors which are not related with the fundamental wave. Those higher harmonic waves are not covered by the FundamentalWave library. It is therefore the user's responsibility to consider these model limitations – with particular focus on the supply conditions.

Any higher harmonic wave effects are also not taken into account by the FundamentalWave library. In case of higher harmonic waves an alternative implementation has to be considered as presented in [17]. The impact of rotor saliency on the fundamental wave components of magnetic flux and magnetic potential (difference) is, however, considered in the presented implementation.

Concentrated windings and fractional slot windings, respectively, are very common in PM synchronous machines due to better field weakening capabilities, higher pole numbers and higher power density. Such fractional slot windings can be considered in the FundamentalWave library, as long as the main power exchanging harmonic component is interpreted as fundamental wave. All higher harmonic waves cause by fractional slot windings are not explicitly considered, but the total effect of those higher harmonics can be taken into account by the total leakage inductance of the stator winding.



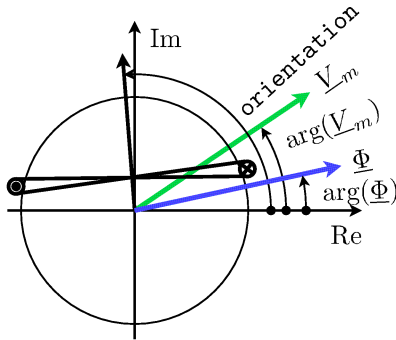


Figure 1: Coil of an equivalent two pole machine with orientation; complex magnetic potential (difference) phasors  $\underline{V}_m$ ; complex magnetic flux phasor  $\underline{\Phi}$

## 3 Electromagnetic Coupling

### 3.1 Single Phase Electromagnetic Coupling

The coupling of fundamental wave magnetic flux and magnetic potential (difference) interacting with instantaneous voltages and currents is elementary modeled in the electro magnetic coupling models `SinglePhaseElectroMagneticConverter`. Figure 1 shows the magnetic phasors and a winding of an equivalent two pole machine. In this case one magnetic pole covers a spatial angle equal to  $\pi$ . The displayed coil is skewed and coil span is smaller than  $\pi$ . The coil can actually also be seen as a distributed winding. Skewing and distributed windings with respect to the fundamental wave are considered by the effective number of turns, represented by the parameter `effectiveTurns`. The effective number of turns of a real machine is determined by the real number of turns, multiplied by the skewing factor and the chording factor. A more detail investigation on common windings and the determination of winding factors is published in [10, 18]. A current through the investigated windings gives rise to magnetic potential (difference) distribution which magnitude is equal effective number of turns times the current. The peak of accessory sinusoidal magnetic potential (difference) caused by the current is in line with the `orientation` of the winding axis.

In an electric machine several windings of stator and rotor windings and permanent magnets contribute the total magnetic potential difference – depending on the type of machine. In the electromagnetic coupling model two physical laws are implemented, i.e., Ampere’s law and the induction law.

### 3.2 Ampere’s Law

Ampere’s law states that the total exciting magneto motive force is equal to the magnetic potential difference. For the investigated single phase winding it is useful to define the complex number of turns:

```
final parameter Complex
N=effectiveTurns*Modelica.ComplexMath.exp(
```

```
Complex(0, orientation))
"Complex number of turns";
```

This complex quantity has the magnitude of the effective number of turns and the phase angle `orientation`. The magnetic potential (difference) of the coupling model,  $v_m$ , and the current of the investigated winding,  $i$ , are then related by:

```
V_m = (2.0/pi)*N*i;
```

The factor  $2.0/\pi$  is the consequence of averaging the sinusoidal fundamental wave flux waveform over one pole pair.

### 3.3 Induction Law

Induction law describes the the relationship between the time derivative of the magnetic flux and the induced voltage of the investigated winding. The projection of the complex magnetic flux onto the `orientation` times the effective number of turns is equal to the negative terminal voltage.

```
-v = Modelica.ComplexMath.real(
Modelica.ComplexMath.conj(N)*
Complex(der(Phi.re),der(Phi.im)));
```

### 3.4 Multi Phase Electromagnetic Coupling

The multi phase electromagnetic coupling model is composed of a vector of  $m$  single phase electromagnetic coupling models, where  $m$  is the number of phases. The  $m$  electrical pins of the single phase electromagnetic coupling model are connected to the  $m$  pins of the electrical multi phases connector used by the multi phase coupling model.

The  $m$  magnetic fundamental wave ports of the single phase electromagnetic coupling models are connected in series. The magnetic series connection is a consequence of, first, each winding being exposed to the same magnetic flux wave – but being located spatially on different locations. Second, the total magnetic potential (difference) excited by all windings is determined by the sum of the magnetic potential (differences) of all individual windings.

## 4 Phase Orientations – Winding Axes

In the `FundamentalWave` library only symmetrical  $m$  phase windings are considered. For multi phase systems and windings with phase numbers greater than three two different cases are distinguished, first, the number of phases is divisible by three and second, the number of phases is not divisible by three. The general function `symmetricOrientation` for determining the orientations of windings of an  $m$  phase electric machine is located in the package `Modelica.Electrical.MultiPhase.Functions`:

```
function symmetricOrientation
extends Modelica.Icons.Function;
input Integer m "Number of phases";
output Modelica.SIunits.Angle
orientation[m]
"Orientation of the resulting
fundamental wave field phasors";
import Modelica.Constants.pi;
```

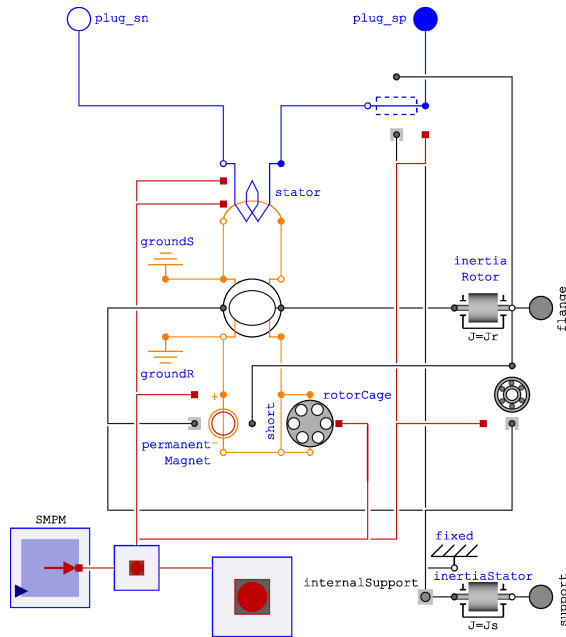


Figure 2: Permanent magnet synchronous machine with optional damper cage

```

algorithm
if mod(m, 2) == 0 then
  // Even number of phases
  if m == 2 then
    // Special case two phase machine
    orientation[1] := 0;
    orientation[2] := +pi/2;
  else
    orientation[1:integer(m/2)] :=
      symmetricOrientation(integer(m/2));
    orientation[integer(m/2) + 1:m] :=
      symmetricOrientation(integer(m/2))
      - fill(pi/m, integer(m/2));
  end if;
else
  // Odd number of phases
  orientation := {(k - 1)*2*pi/m
    for k in 1:m};
end if;
symmetricOrientation;
    
```

So the function is designed recursively so that subsystems are modularly designed. In the following some examples of phase numbers will be discussed. Two, four, eight, sixteen, thirty-two, etc. phase windings are currently not supported, as in general phase numbers equal to  $2^n$  with integer  $n$  are not considered.

In order to summarize mathematical equations for different phase numbers  $m$  in the following, abbreviation  $\text{orientation}_k$  will be used to indicate the angles of the orientation of the winding axes of a symmetrical  $m$  phase winding. So  $\text{orientation}_k$  is the  $k$ -th element (phase index) of the result vector returned by function `symmetricOrientation`, called with argument  $m$ .

The symmetrically supply voltages and currents, respectively, have the phase angles

$$\phi_k = -\text{orientation}_k$$

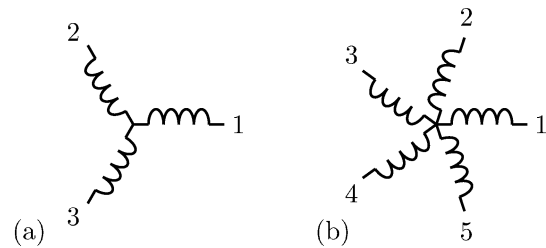


Figure 3: Winding axes of symmetrical (a) three phase winding and (b) five phase winding

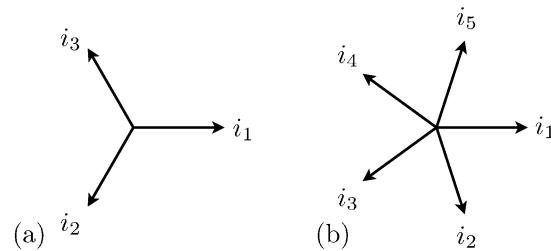


Figure 4: Symmetrical phase angles of voltages and currents, respectively, of (a) three phase winding and (b) five phase winding

The winding orientations and the phase shifts of the supplying system have the same magnitudes for each phase index  $k$ , but different signs. This is a general property of symmetrical systems supplying symmetrical windings, see Figs. 3–6.

In the following only symmetrical winding axes will be assumed. The phase angles of symmetrical voltage and current supply are also presented, even though symmetric supply is not assumed in the FundamentalWave library.

#### 4.1 Odd Phase Numbers

For all odd phase numbers  $m$  (not divisible by 2) the symmetrical orientations of the winding axes are

$$\text{orientation}_k = \frac{2\pi(k-1)}{m}.$$

So this applies for the case  $m = 3$ ,  $m = 5$ ,  $m = 7$ ,  $m = 9$ ,  $m = 11$ ,  $m = 13$ ,  $m = 15$ , etc., see Fig. 3–4.

#### 4.2 Even Phase Numbers

For even phase numbers unequal to  $2^n$  with integer  $n$ , the  $m$  phase system is separated into two subsystems with  $m/2$  phases. The winding orientations of the second sub system lags the first sub system by  $\frac{\pi}{m}$ . This is then the point where the recursive determination of phase angle is initiated. For each of the two sub systems functions `symmetricOrientation` is called, considering the lag angle  $\frac{\pi}{m}$ . It is important to explicitly note that the phase shift between the two sub systems is not  $\frac{2\pi}{m}$ , since in this case the

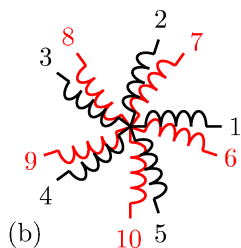
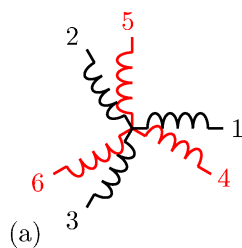


Figure 5: Winding axes of symmetrical (a) six phase winding and (b) ten phase winding

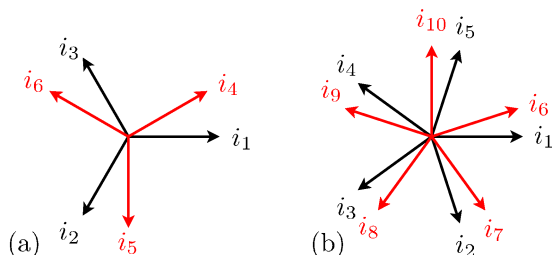


Figure 6: Symmetrical phase angles of voltages and currents, respectively, of (a) six phase winding and (b) ten phase winding

two sub systems where aligned at  $\pm\pi$  which does not make sense in a technical system for redundancy reasons.

A **six phase system** is then separated into two three phase systems. The winding orientations of the second sub systems lags the first sub system by  $\pi/6$ , see Figs. 5–6. The phase sequences (1-2-3) and (4-5-6) of the two sub systems are equal.

A **ten phase system** consists of two five phase systems with the phase sequences (1-2-3-4-5) and (6-7-8-9-10). The second sub systems lags the first sub system by  $\frac{\pi}{10}$ , see Figs. 5–6.

### 4.3 Phase Numbers Divisible by Three

Phase numbers divisible by three are either covered by subsection 4.1 and 4.2. Therefore, from a formal point of view no additional explanations are required to handle, for example, nine phase machines. Yet, a typical engineering approach and function `symmetricOrientation` for numbering the phase numbers are different and may require some additional comments:

Practically, in most technical cases, electrical machines with phase numbers divisible by three will be supplied by an appropriate number of three phase inverters. For six phase systems this has already been demonstrated in subsection 4.2. In the FundamentalWave library nine phase systems are handled differently only in that sense, the sequence numbering the phase windings is most likely different from an engineer who uses three three phase inverters. In the engineering phases (1-2-3) are most likely assigned to the first inverter, phases (4-5-6) are assigned to the second inverter and phases (7-8-9) are assigned to the third inverters, see Fig. 7(a). In the FundamentalWave library the phase are

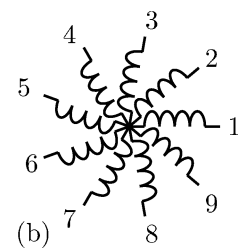
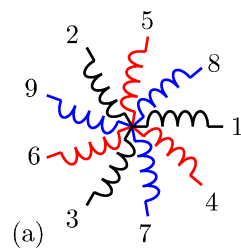


Figure 7: Numbering of nine phase symmetrical winding according to (a) an engineering approach using three three phase inverters and (b) the scheme of the FundamentalWave library

numbered according to Fig. 7(b). Even though the numbering is different the angles of the orientations are fully identical.

For a fifteen phase a design engineer could always argue whether such system can be seen as five three phase systems or as three five phase systems. However, from operating conditions point of view, there is no difference between these two cases. The numbering scheme of the FundamentalWave follows a formal scheme and the user decides how the machine phases are supplied.

## 5 Magnetic Components

All the existing magnetic components of the FundamentalWave library can be re-used for the multi-phase machine models, since the magnetic port representation did not change. In the current implementation only linear magnetic materials are considered. Saturation effects are not taken into account.

In all electric machine models of the FundamentalWave library the total magnetic reluctance is concentrated in the air gap model. An example of permanent magnet synchronous machine with optional damper cage is displayed in Fig. 2. In the actual implementation of the FundamentalWave library the magnetic reluctances of the stator, rotor and air gap are not individually modeled. Even the linear characteristic of the permanent magnet is represent by the total air gap reluctance of the machine. The total reluctance takes the variable reluctance of the air gap length  $\delta$  into account. A sketch of the air gap and the reciprocal function  $1/\delta$  are shown in Fig. 8.

The effect of variable magnetic reluctance due to the uneven shape of the air gap and the arrangement of magnets, respectively, is called saliency. The effect of saliency on fundamental wave forms is fully considered by unequal direct ( $d$ ) and quadrature ( $q$ ) axis reluctances. For rotor fixed magnetic flux  $\Phi_{\text{m}}$  and total magnetic potential difference  $v_{\text{m}}$  the following relationship applies:

$$\begin{aligned} (\pi/2.0)*V_{\text{m.re}} &= \Phi_{\text{m.re}} * R_{\text{m.d}}; \\ (\pi/2.0)*V_{\text{m.im}} &= \Phi_{\text{m.im}} * R_{\text{m.q}}; \end{aligned}$$

The  $d$  and  $q$  axis are, however, fixed with the rotor structure. Magnetic rotor excitation of synchronous machines is, however, always aligned with the  $d$  axis.

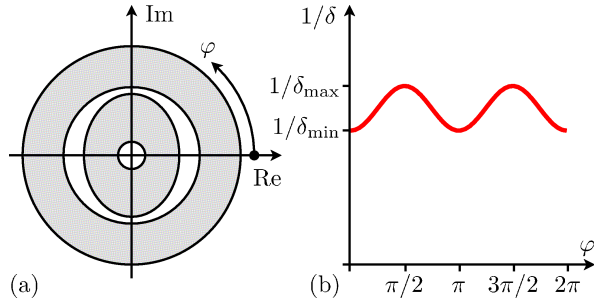


Figure 8: (a) Variable air gap length  $\delta$  of a synchronous machine and (b) reciprocal air gap function  $1/\delta$  versus spatial angle  $\varphi$  of an equivalent two pole machine

Quantity	$m_s = 3$	$m_s \geq 3$
Nominal stator phase voltage	$V'_{sN}$	$V_{sN} = V'_{sN}$
Nominal stator phase current	$I'_{sN}$	$I_{sN} = I'_{sN} \frac{3}{m}$
Nominal stator frequency	$f'_{sN}$	$f_{sN} = f'_{sN}$
Nominal electrical torque	$\tau'_N$	$\tau_N = \tau'_N$
Nominal electrical stator power	$P'_{sN}$	$P_{sN} = P'_{sN}$

Table 1: Parameters of machines with phases numbers equal and greater than three

In case that the saturation characteristics of the different regions of the machine shall be considered in the future, the magnetic equivalent circuit has to be adapted such way that each region is then represented by one non-linear reluctance.

## 6 Parametrization

Where do the parameters of a machine with  $m_s$  stator phases come from? First, in the design stage of the machine, an engineer determines these parameters from finite element analysis or any other electromagnetic design software. Second, the parameters of a three phase machine are known or estimated and the users wants to determine the parameters of an equivalent  $m$  phase machine. The equivalence then often refers to equivalent speed, frequency, torque, power, phase voltage, power factor and efficiency. For the second case the exact calculations will be provided in the following:

Assume, the nominal parameter of a three phase and arbitrary  $m$  phase machine as listed in Tab. 1. All the parameters of a three phase machine are indicated with '. According to the relationship between the  $m$  phase nominal phase voltage and current, all resistances and inductances of an  $m$  phase machine are scaled with  $m/3$ . A list of relevant parameters is summarized in Tab. 2.

The rotor winding of squirrel cage induction machines are implemented as equivalent  $m_s$  phase windings – where  $m_s$  is the number of stator phases. Slip ring induction machines may have different phases numbers of stator and rotor – where  $m_r$  is the number of rotor phases. For synchronous machines with permanent magnets, electrical excitation and reluctance rotor, the optional damper cage is implemented

Quantity	$m = 3$	$m \geq 3$
Stator resistance	$R'_s$	$R_s = R'_s \frac{m}{3}$
Stator stray inductance	$L'_{s\sigma}$	$L_{s\sigma} = L'_{s\sigma} \frac{m}{3}$
Main field inductance	$L'_m$	$L_m = L'_m \frac{m}{3}$
Main field inductance, $d$ -axis	$L'_{md}$	$L_{md} = L'_{md} \frac{m}{3}$
Main field inductance, $q$ -axis	$L'_{mq}$	$L_{mq} = L'_{mq} \frac{m}{3}$

Table 2: Stator parameters of three and  $m_s \geq 3$  phase machines

Quantity	$m = 3$	$m \geq 3$
Induction machine with squirrel cage		
Rotor cage resistance	$R'_r$	$R_r = R'_r \frac{m_s}{3}$
Rotor stray inductance	$L'_{r\sigma}$	$L_{r\sigma} = L'_{r\sigma} \frac{m_s}{3}$
Induction machine with slip ring rotor		
Rotor cage resistance	$R'_r$	$R_r = R'_r \frac{m_r}{3}$
Rotor stray inductance	$L'_{r\sigma}$	$L_{r\sigma} = L'_{r\sigma} \frac{m_r}{3}$
All synchronous machines		
Damper cage resistance, $d$ -axis	$R'_{rd}$	$R_{rd} = R'_{rd}$
Damper cage resistance, $q$ -axis	$R'_{rq}$	$R_{rq} = R'_{rq}$
Damper cage stray inductance, $d$ -axis	$L'_{r\sigma,d}$	$L_{r\sigma,d} = L'_{r\sigma,d}$
Damper cage stray inductance, $q$ -axis	$L'_{r\sigma,q}$	$L_{r\sigma,q} = L'_{r\sigma,q}$

Table 3: Rotor parameters of machines with three and  $m_s \geq 3$  and  $m_r \geq 3$  phases

with salient rotor parameters with respect to the direct ( $d$ ) and quadrature ( $q$ ) axis. So there is no difference between three and multi phase damper cage models and parameters. The rotor cage parameters of all machine types are summarized in Tab. 3.

## 7 Examples

In the FundamentalWave library there are examples for all types of machines, comparing three phase and multi phase ( $m = 5$ ) machines. The three and five phase machines are operated with equal nominal phase voltages. The parameters of the five phase machine are parameterized such way automatically that the phase number can be increased without changing torques and powers for the multi phase machines. In a duplicate example the phase numbers can be changed for from five to higher numbers.

The following examples are included in the FundamentalWave library:

### 7.1 Induction Machine with Squirrel Cage

In model `Examples.AIMC_DOL_MultiPhase` two asynchronous induction machines with squirrel cage rotor are started directly on line (DOL) by means of an ideal switch; see Figure 9. The machines start from standstill.

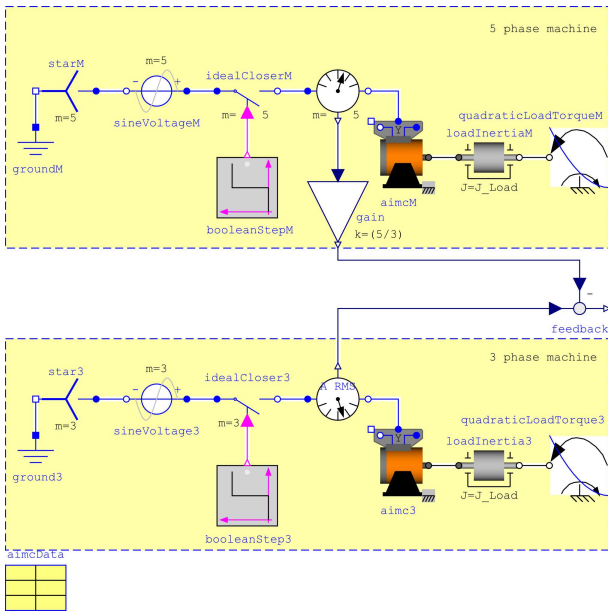


Figure 9: Comparing a three and a multi phase ( $m = 5$ ) permanent magnet synchronous machine, operated on an idealized voltage inverter

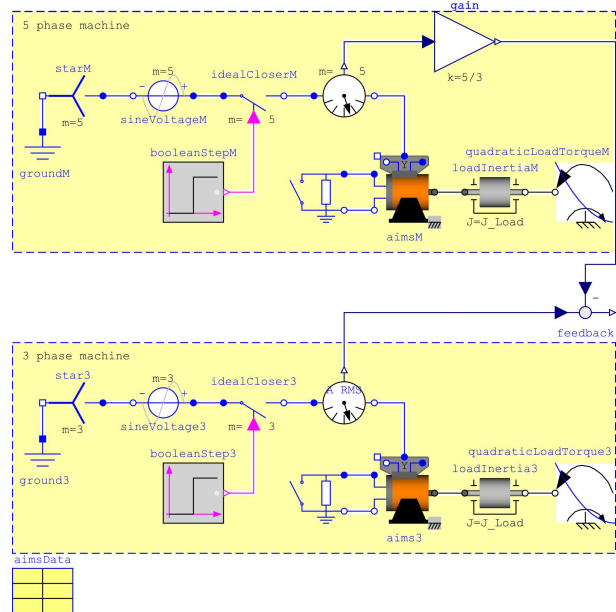


Figure 12: Comparing a three and a multi phase ( $m = 5$ ) induction machine started directly on line, operated on an idealized voltage inverter

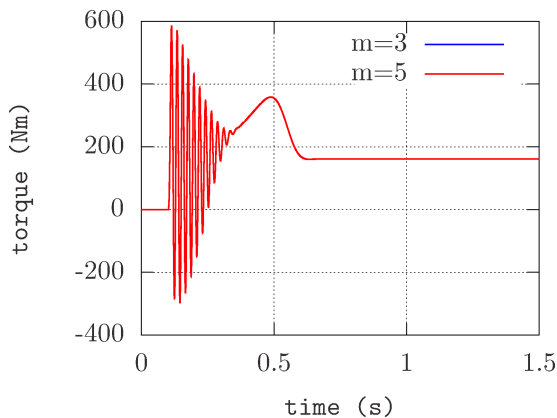


Figure 10: Simulation result of electrical torque of a three and five phase squirrel cage induction machines; the torques are equal

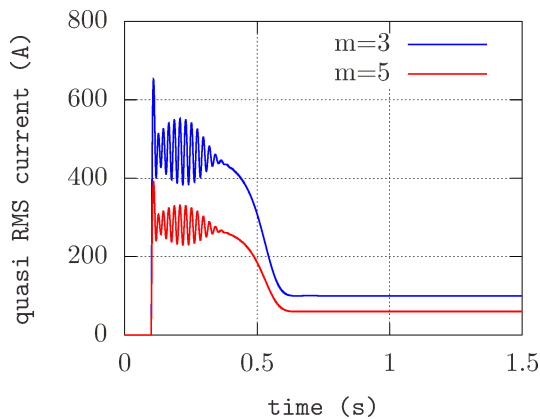


Figure 11: Simulation result of the quasi RMS currents of a three and five phase squirrel cage induction machines; the current ratio is five to three

The mechanical load is modeled by means of a quadratic speed dependent load torque and an additional load inertia. This example demonstrates equivalent dynamic behavior of the three and five phase machine. Particularly, the electrical torque, speed, and the particular losses are equal.

Both machines have the same nominal phase voltage, but different nominal phase currents according to Tab. 1. In Fig. 10 the two identical electric torques of the two machines are shown. The different quasi RMS currents of two machines are displayed in Fig. 11. The current ratio is equal to five over three.

## 7.2 Induction Machine with Slip Ring Rotor

Model Examples.BasicMachines.AIMS\_Start\_MultiPhase compares a three and a five phase slip ring induction machine, operating the stator direct on line; see Fig. 12 The number of stator phases  $m_s = 5$  and the number of rotor phases,  $m_r = 5$ , are equal. The multi phase rotor windings of each machine are connected with a rheostat which is shorted after a give time period  $t_{Rheostat} = 1.0$  second. The rheostat enables a greater starting torque – but worse efficiency. Therefore, after one second, the rheostats are shortened to achieve a higher efficiency and speed of the machines. The user can copy the example and change the rotor phase number  $m_r$  such way that it differs from the stator phase number,  $m_s$ . This case is also supported the FundamentalWave library. In Fig. 13 and 14 the electromagnetic torques and the quasi RMS currents of the two machines are compared. The torques are identical and the current ratio is five to three according to Tab. 1.

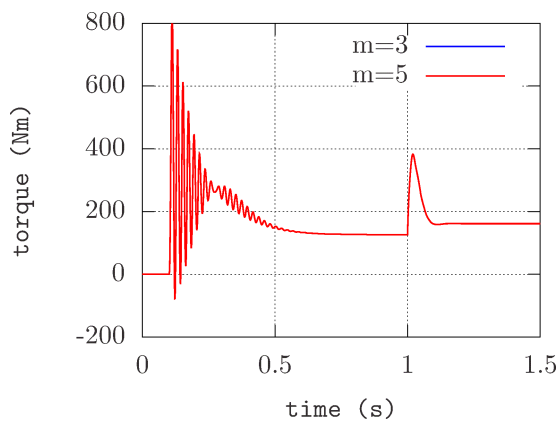


Figure 13: Simulation result of electrical torque of a three and five phase slip ring induction machines; the torques are equal

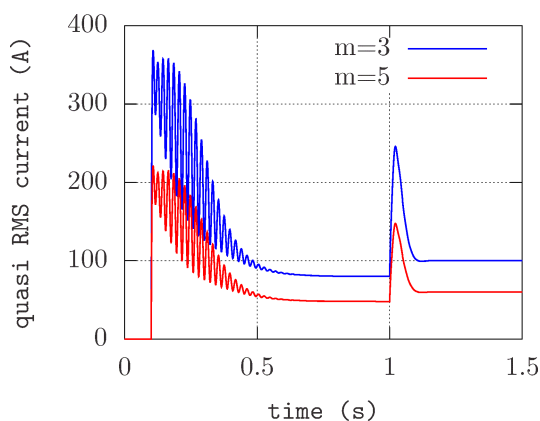


Figure 14: Simulation result of the quasi RMS currents of a three and five phase slip ring induction machines; the current ratio is five to three

### 7.3 Synchronous Generator with Electrical Excitation

In example `Examples.SMPM_Generator` two mains supplied electrical excited synchronous machine with three and five stator phases are compared; see Fig. 16. For each machine shaft speed is constant and slightly different than synchronous speed. In this experiment each rotor is forced to make a full revolution relative to the magnetic field. In Fig. 16 the generated torques versus load angle are shown for a fixed level of excitation. In addition to the sinusoidal waveform of the torque a second harmonic component is superimposed due to the saliency of the rotor. However, for the investigated machines, the saliency effect is very small so that the torque waveform almost appears as a pure sine wave. The quasi RMS currents of the two machines are compared in Fig. 17. The current ratios of the three and five phase machine is five to three.

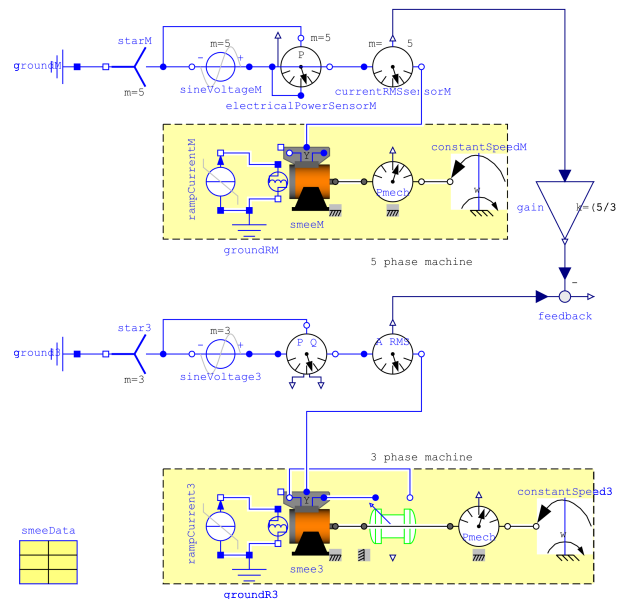


Figure 15: Comparing a three and a multi phase ( $m = 5$ ) permanent magnet synchronous machine, operated on an idealized voltage inverter

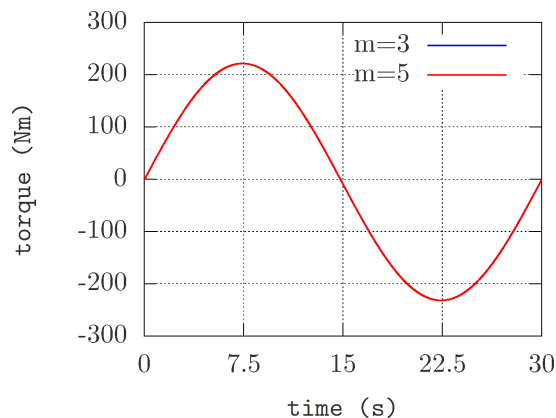


Figure 16: Simulation result of electrical torque, comparing a three and five phase

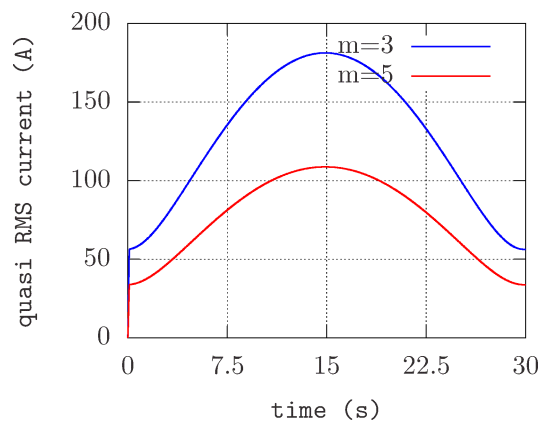


Figure 17: Simulation result of the quasi RMS currents of a three and five phase synchronous machine with electric excitation; the current ratio is five to three

## 8 Conclusions

The paper presents an extension of the FundamentalWave library towards multi phase stator (and rotor) windings with phase numbers greater or equal than three. This library is included in the MSL 3.2.1. Assumptions and limitations of the presented implementation are explained. In the new FundamentalWave library only symmetrical windings are supported. The structures of symmetrical multi phase windings and supplies are introduced.

The parametrization of the multi phase machines is discussed. Conversion tables for parameterizing multi phase machines equivalent to three phase machines are presented. Simulation examples of three and equivalent five phase induction and synchronous machines are presented and compared.

## 9 Acknowledgement

The research leading to these results has received funding from the ENIAC Joint Undertaking under grant agreement no. 270693-2 and from the Österreichische Forschungsförderungsgesellschaft mbH under project no. 829420.

## References

- [1] B. Stumberger, G. Stumberger, A. Hamler, M. Trelep, M. Jesenik, and V. Gorican, "Increasing of output power capability in a six-phase flux-weakened permanent magnet synchronous motor with a third harmonic current injection," *IEEE Transactions on Magnetics*, vol. 39, pp. 3343–3345, 2003.
- [2] D. G. Dorrell, C. Y. Leong, and R. A. McMahon, "Analysis and performance assessment of six-pulse inverter-fed three-phase and six-phase induction machines," *IEEE Transactions on Industry Applications*, vol. 42, pp. 1487–1495, November/December 2006.
- [3] G. Aroquiadassou, A. Mpanda-Mabwe, F. Betin, and G.-A. Capolino, "Six-phase induction machine drive model for fault-tolerant operation," *SDEMPED*, 2009.
- [4] H. A. Toliyat, M. M. Rahimian, and T. A. Lipo, "dq modeling of five phase synchronous reluctance machines including third harmonic of air-gap mmf," *Conference Record of the 1991 IEEE Industry Applications Society Annual Meeting, 1991.*, pp. 231–237, 1991.
- [5] D. Dujic, M. Jones, and E. Levi, "Features of two multi-motor drive schemes supplied from five-phase/five-leg voltage source inverters," *Conference Proceedings on Power Conversion and Intelligent Motion, PCIM, Nuremberg, Germany*, no. S2d-2, 2008.
- [6] D. Dujic, M. Jones, and E. Levi, "Analysis of output current ripple rms in multiphase drives using space vector approach," *IEEE Transactions on Power Electronics*, vol. 24, no. 8, pp. 1926–1938, 2009.
- [7] R. L. A. Ribeiro, C. B. Jacobina, A. M. N. Lima, and E. R. C. da Silva, "A strategy for improving reliability of motor drive systems using a four-leg three-phase converter," *Sixteenth Annual IEEE Applied Power Electronic Conference and Exposition. APEC 2001*, vol. 1, pp. 385–391, 2001.
- [8] R. Errabelli and P. Mutschler, "Fault-tolerant voltage source inverter for permanent magnet drives," *IEEE Transactions on Power Electronics*, vol. 27, pp. 500–508, feb. 2012.
- [9] T. Treichl, *Regelung von sechssträngigen permanentregten Synchronmaschinen für den mobilen Anwendungsfall*. PhD thesis, FernUniversität, 2006.
- [10] C. Kral and A. Haumer, "Modelica libraries for DC machines, three phase and polyphase machines," *International Modelica Conference, 4th, Hamburg, Germany*, pp. 549–558, 2005.
- [11] C. Kral and A. Haumer, *Object Oriented Modeling of Rotating Electrical Machines*. INTECH, 2011.
- [12] C. Kral and A. Haumer, "The new fundamentalwave library for modeling rotating electrical three phase machines," *8th International Modelica Conference*, 2011.
- [13] T. A. Lipo, "A d-q model for six phase induction machines," *Conference Record of International Conference on Electrical Machines, ICEM*, pp. 860–867, 1980.
- [14] E. Andresen and K. Bieniek, "Der Asynchronmotor mit drei und sechs Wirkungssträngen am stromeinprägenden Wechselrichter," *Archiv für Elektrotechnik*, vol. 63, pp. 153–167, 1981.
- [15] H.-H. Jahn and R. Kasper, "Koordinatentransformationen zur Behandlung von Mehrphasensystemen," *Archiv für Elektrotechnik*, vol. 56, pp. 105–111, 1974.
- [16] J. Stepina, *Einführung in die allgemeine Raumzeiger-Theorie der elektrischen Maschinen*. Kaiserslautern: Vorlesungsskriptum, Universität Kaiserslautern, 1979.
- [17] C. Kral, A. Haumer, M. Bogomolov, and E. Lomonova, "Harmonic wave model of a permanent magnet synchronous machine for modeling partial demagnetization under short circuit conditions," *XXth International Conference on Electrical Machines (ICEM)*, pp. 295–301, Sept. 2012.
- [18] C. Kral and A. Haumer, "Simulation of electrical rotor asymmetries in squirrel cage induction machines with the extendedmachines library," *International Modelica Conference, 6th, Bielefeld, Germany*, no. ID140, pp. 351–359, 2008.





# New Multi Phase Quasi Static Fundamental Wave Electric Machine Models for High Performance Simulations

Christian Kral    Anton Haumer  
Electric Machines, Drives and Systems    Technical Consulting  
1060 Vienna, Austria    3423 St.Andrä-Wördern, Austria  
dr.christian.kral@gmail.com    a.haumer@haumer.at

## Abstract

A new quasi static fundamental wave machines library will be included in the magnetic domain package of the next Modelica Standard Library (MSL). The provided classes of machine models omit all transient electrical effects, but mechanical dynamics are fully taken into account. By including the new machine models new classes of problems can be treated, enabling fast electric machine and drive simulations. Yet, all the characteristic loss effects of transient machine models are fully taken into account, where needed. Phase numbers greater than three are supported. For each machine type available in the MSL there will then exist both a fully transient and a quasi static electric machine model. The package structure of the quasi static fundamental wave package and the concept of implementation will be presented. All required assumptions and limitations for operating the new machine models will be presented and discussed. Deviating parameters compared to the transient machine models will be discussed and explained. Simulation examples will be presented and compared with transient simulation experiments. Possible applications for the new machine models will be outlined.

*Keywords: Quasi static fundamental wave electric machine models, multi phase, transient effects, reference frame*

## 1 Introduction

The investigation of electric machine transients is highly relevant for designing drive controls and investigating complex multi domain physical systems. Yet, in many applications the full electrical transients play a minor role. For example, the investigation of electric driving cycles of full or hybrid electric vehicles or railways or the assessment of auxiliary drives or the superior control of multiple drives in an industrial environment do not require the full consideration of all electrical transients. In such cases it is assumed that machine control is designed and implemented such way that electric transients have no significant impact either on power and energy balance or on the overloading capability or stability. Additionally, it may even be requested or required to reduce system complexity for such applications

by eliminating electrically transient phenomena. The newly developed quasi static fundamental wave electric machines library is closing the gap for fast, simple and accurate electric machine models in a Modelica simulation environment. The new quasi static fundamental wave electric machines library is designed in the style of the transient FundamentalWave library [1] which is already included in the Modelica Standard Library (MSL). The new library utilizes the existing quasi static multi phase electric package as well as the rotational mechanics and the thermal domain [2]. All machine models are strictly object oriented: Modelica classes for windings, air gaps, linear magnetic reluctances, losses, etc. are provided. From a didactic point of view the quasi static magnetic field theory enables valuable insight into electro magnetic and mechanic power conversion. Additionally, the provided linear machine models are designed such way that they can be extended towards non linear magnetic effects such as saturation or permanent magnet (PM) demagnetization. When designing the new library it was intended to include as much features as provided by the transient machine models. Therefore, phase numbers greater than three and different phase numbers in the stator and the slip ring rotor of induction machines are fully supported. Yet the number of phases,  $m$ , being equal to  $2^n$  with integer  $n$  are currently not included. The reason for omitting such phase numbers is caused by the fundamental difference of windings systems with  $m = 2^n$  from other windings. An alternative concept for modeling quasi static magnetic machines in Modelica is presented in [3]. In this concept an induction machine model is presented based solely on electric and magnetic quasi static equivalent circuits. Electro mechanical power conversion is not yet included in this proposal.

## 2 Connector Concept

From a formal point of view the connector concept of the transient and the quasi static fundamental wave package look very similar. The magnetic port consists of the complex magnetic potential,  $\underline{V}_m = V_{m, \text{re}} + jV_{m, \text{im}}$ , and the complex magnet flux,  $\underline{\Phi} = \Phi_{\text{re}} + j\Phi_{\text{im}}$ .

```
connector MagneticPort
  "Basic quasi static magnet connector"
```

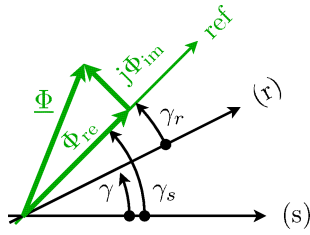


Figure 1: Stator and rotor fixed reference frame of an electric machine

```

Modelica.SIunits.ComplexMagneticPotential
  V_m "Complex magnetic potential
    at the node";
  flow Modelica.SIunits.ComplexMagneticFlux
    Phi "Complex magnetic flux flowing
    into the pin";
end MagneticPort;
    
```

The complex magnetic potential and the complex magnetic flux represent a spatial fundamental wave field distribution given by:

$$\begin{aligned}\Phi(\varphi) &= \text{Re}[(\Phi_{\text{re}} + j\Phi_{\text{im}})e^{-j\varphi}] \\ &= \Phi_{\text{re}} \cos(\varphi) + \Phi_{\text{im}} \sin(\varphi) \\ V_m(\varphi) &= \text{Re}[(V_{m,\text{re}} + jV_{m,\text{im}})e^{-j\varphi}] \\ &= V_{m,\text{re}} \cos(\varphi) + V_{m,\text{im}} \sin(\varphi)\end{aligned}$$

A visualization of the fundamental wave forms is illustrated in [1].

The main difference is the reference angle included in both the quasi static positive and negative connector class definition.

```

connector PositiveMagneticPort
  "Positive magnetic port"
  extends QuasiStaticFundamentalWave.
    Interfaces.MagneticPort;
  Modelica.Electrical.QuasiStatic.
    Types.Reference reference "Reference";
end PositiveMagneticPort;
    
```

The reference angle represents the angle of the reference frame that the connector refers to. In electric machines typically stator and rotor reference frames are used in which the fundamental principles of Ampere's law and induction law apply. The stator of electric motors are either supplied by a fixed or variable frequency source. Stator frequency determines the stator reference angle. The frequency of the induced voltages of the rotor depends on the machine type and whether the winding is accessible from the outside. In case of a synchronous generator stator frequency is determined by the rotational speed.

Stator and rotor reference frames with respect to a complex magnetic flux phasor are depicted in Fig. 1. The complex phasor consists of a real and imaginary part, in the respective reference frame. The different reference frames are, however, indicated by the reference angle provided by the connector. So when coupling the stator and rotor fixed reference frame over the air gap model the real and imaginary

parts of magnetic flux are the same. Only the reference angles are different. The very same applies for the complex magnetic potential difference.

In Fig. 1 (s) represents the stator fixed reference frame and  $\gamma_s$  is the angle difference between connector reference frame and the stator fixed reference frame. In the same way, (r) represents the rotor fixed reference frame and  $\gamma_r$  is the angle difference between connector reference frame and the rotor fixed reference frame. The angle difference

$$\gamma = \gamma_s - \gamma_r$$

is equivalent to the rotational mechanical angle between stator and rotor, multiplied by the number of pole pairs,  $p$ .

### 3 Assumptions and Limitations

First, in the actual version of the quasi static fundamental wave library the phase number  $m$  must not be equal to  $2^n$  for integer  $n$ .

Second, the used electric connector is based on the package `Modelica.Electrical.QuasiStatic.MultiPhase`. Therefore, only single frequency voltages and currents are taken into account in the underlying electrical connector concept.

The third limitation is that only symmetric voltages and currents are allowed for supplying and loading the machines. This restriction is a consequence of the fact that unbalanced voltage and current supply, respectively, cause forward and backward spinning magnetic field waves. In the squirrel cages of induction machines the backwards spinning field waves give rise to slip dependent frequency components in the voltages and currents which cannot be taken into account by the single supply frequency approach of the quasi static multi phase connector. When investigating the power flow and systemic behaviors of mains supplied electric machines any supply asymmetrical is usually of minor interest. For variable speed inverter fed electric machines symmetrical voltage supply can be assumed due to the control of the power electronics which strictly avoids voltage asymmetries.

Fourth, all windings are assumed to be fully symmetrical. So the numbers of turns are equal for all winding axes and the winding axis orientations are strictly related with the function shown in Listing 1.

Fifth, due to the assumed symmetry of supply voltages and currents, zero voltages, zero currents and zero impedances are not considered in the quasi static machine models.

Sixth, all the magnetic reluctances of the machine models are assumed to be constant. This represents a strictly linear relationship between magnetic potential differences and magnetic fluxes.

### 4 Library Structure and Components

The structure of the new quasi static fundamental wave library is presented in Fig. 2. The key components such as

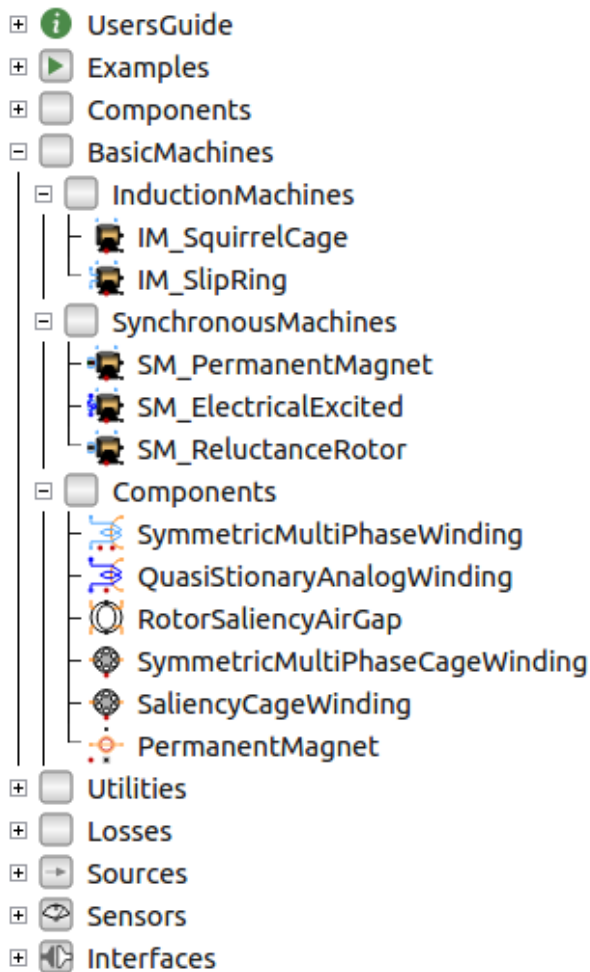


Figure 2: Structure of the quasi static fundamental wave library

the electro magnetic coupling, the salient air gap model and the damper cage concept which will be presented in the following subsections. The coupling models rely on the symmetry of voltages and currents. Therefore, symmetrical components are discussed in this section as well. Cage models, the PM model, regular reluctance and eddy current models are designed in the style of the transient fundamental wave library and need no particular attention in this paper. More detailed model descriptions can be found in [1]. Parametrization rules for multi phase machines with phase numbers greater or equal to three are described in [4].

## 4.1 Symmetrical Components

The orientations of the winding axes of an  $m$ -phase system is defined by the function listed in Listing 1, which is also used in the transient fundamental wave library.

Listing 1: Function symmetricOrientation

```
function symmetricOrientation
  "Orientations of the resulting
  fundamental wave field phasors"
  extends Modelica.Icons.Function;

  input Integer m "Number of phases";
```

```
output Modelica.SIunits.Angle
orientation[m]
"Orientation of the resulting
fundamental wave field phasors";

import Modelica.Constants.pi;

algorithm
  if mod(m, 2) == 0 then
    // Even number of phases
    if m == 2 then
      // Special case two phase machine
      orientation[1] := 0;
      orientation[2] := +pi/2;
    else
      orientation[1:integer(m/2)] :=
        symmetricOrientation(integer(m/2));
      orientation[integer(m/2) + 1:m] :=
        symmetricOrientation(integer(m/2))
        - fill(pi/m, integer(m/2));
    end if;
  else
    // Odd number of phases
    orientation :=
      {(k - 1)*2*pi/m for k in 1:m};
  end if;
end symmetricOrientation;
```

An arbitrary  $m$ -phase system of currents  $i_{[k]}$ , for  $1 \leq k \leq m$ , can be transformed into  $m$  symmetrical components. In case of a fully symmetrical system of currents, only the positive sequence component is non-zero. The positive sequence component is computed by means of multiplying the current vector with the transformation matrix obtained from function `symmetricOrientationMatrix`. The design of the transformation matrix relies on the recursive application of the function presented in Listing 1.

As a consequence of assuming fully symmetrical voltages and currents, respectively, only the positive sequence of voltages and currents arise. In case of even phase numbers more than one positive sequence component will arise. It is thus required to determine the indexes of positive sequence components by means of function `indexPositiveSequence`. All other symmetrical components are equal to zero. Their indexes are determined by function `indexNonPositiveSequence`.

## 4.2 Electromagnetic Coupling

The induction law and Ampere's law are related with the positive sequences of the symmetrical components of voltages and currents. Due to the symmetry of the windings the induced voltages of all positive sequence components are identical. In the quasi static domain the time derivative of the magnetic flux is replaced by a multiplication with the imaginary unit and the angular frequency. The complex total magnetic potential difference is related with the sum of all positive sequence currents, see Listing 2.

Listing 2: Electromagnetic coupling model incorporating the induction law and Ampere's law

```
model MultiPhaseElectroMagneticConverter
  "Multi phase electro magnetic converter"
  ...
  QuasiStationary.MultiPhase.Interfaces.
```

```

PositivePlug
plug_p(final m=m) "Positive plug";
QuasiStationary.MultiPhase.Interfaces.
NegativePlug
plug_n(final m=m) "Negative plug";
Interfaces.PositiveMagneticPort port_p
"Positive complex magnetic port";
Interfaces.NegativeMagneticPort port_n
"Negative complex magnetic port";
Modelica.SIunits.ComplexVoltage
v[m] "Voltage drop";
Modelica.SIunits.ComplexCurrent
i[m] "Current";
...
SIunits.ComplexVoltage
vSymmetricalComponent[m] =
symmetricTransformationMatrix(m)*v
"Symmetrical components of voltages";
SIunits.ComplexCurrent
iSymmetricalComponent[m] =
symmetricTransformationMatrix(m)*i
"Symmetrical components of currents";
protected
final parameter Integer indexNonPos[:]=
indexNonPositiveSequence(m)
"Indices of all non positive sequence
components";
final parameter Integer indexPos[:]=
indexPositiveSequence(m)
"Indices of all positive sequence
components";
equation
// Magnetic flux and flux balance
// of the magnetic ports
port_p.Phi = Phi;
port_p.Phi + port_n.Phi = Complex(0,0);
// Magnetic potential difference
// of the magnetic ports
port_p.V_m - port_n.V_m = V_m;
// Voltage drop of electrical plugs
v = plug_p.pin.v - plug_n.pin.v;
// Current and current balance of plugs
i = plug_p.pin.i;
plug_p.pin.i + plug_n.pin.i =
{Complex(0,0) for k in 1:m};
// Amperes law
V_m.re = sqrt(2) * (2.0/pi) *
Modelica.ComplexMath.real(
N*iSymmetricalComponent[1])*m/2;
V_m.im = sqrt(2) * (2.0/pi) *
Modelica.ComplexMath.imag(
N*iSymmetricalComponent[1])*m/2;
for k in 1:size(indexNonPos,1) loop
iSymmetricalComponent[indexNonPos[k]] =
Complex(0,0);
end for;
// Induction law
for k in 2:size(indexPos,1) loop
vSymmetricalComponent[indexPos[1]] =
vSymmetricalComponent[indexPos[k]];
end for;
-sqrt(2) * Complex(
Modelica.ComplexMath.real(
vSymmetricalComponent[indexPos[1]]),
Modelica.ComplexMath.imag(
vSymmetricalComponent[indexPos[1]]))
= Modelica.ComplexMath.conj(N)*j*omega*Phi;
// Breakable connections of references
Connections.branch(
port_p.reference, port_n.reference);
port_p.reference.gamma =
port_n.reference.gamma;
Connections.branch(

```

```

plug_p.reference, plug_n.reference);
plug_p.reference.gamma =
plug_n.reference.gamma;
Connections.branch(
plug_p.reference, port_p.reference);
plug_p.reference.gamma =
port_p.reference.gamma;
...
end MultiPhaseElectroMagneticConverter;

```

The reference angles of both the electrical and the magnetic domain are connected by means of breakable connections. This concept breaks potential algebraic loops of the electric and magnetic domain. Additionally, both domains are interconnected by a breakable connector as well.

### 4.3 Air Gap

The air gap model consists of two stator and two rotor magnetic ports and two rotational mechanic flanges representing the stator and rotor, respectively. The complex magnetic potential difference and flow quantities of the quasi static fundamental wave stator and rotor connectors are identical. However, the reference angles are different according to Fig. 1.

Rotor saliency is represent by different magnetic reluctances in the  $d$  (direct) and  $q$  (quadrature) axis – with respect to the rotor fixed reference frame. Therefore, the relationships between magnetic potential differences and fluxes have to be expressed in the rotor fixed reference frame: the rotor fixed complex magnetic potential differences and fluxes are obtained by multiplying each of these quantities by  $e^{j\gamma}$  as shown in Listing 3.

Listing 3: Air gap model with rotor saliency

```

model RotorSaliencyAirGap "Air gap model
with rotor saliency"
Interfaces.PositiveMagneticPort
port_sp "Positive complex magnetic
stator port";
Interfaces.NegativeMagneticPort
port_sn "Negative complex magnetic
stator port";
Interfaces.PositiveMagneticPort port_rp
"Positive complex magnetic rotor port";
Interfaces.NegativeMagneticPort port_rn
"Negative complex magnetic rotor port";
Rotational.Interfaces.Flange_a flange_a
"Flange of the rotor";
Rotational.Interfaces.Flange_a support
"Support at which the reaction torque
is acting";
parameter ...
// Phasors of magn. potential differences
SIunits.ComplexMagneticPotentialDifference
V_ms
"Complex magnetic potential difference
of stator w.r.t. stator ref. frame";
SIunits.ComplexMagneticPotentialDifference
V_msr = V_ms *
ComplexMath.fromPolar(1,gammar)
"Complex magn. potential difference of
stator w.r.t. rotor fixed ref. frame";
...
Modelica.SIunits.Torque tauElectrical
"Electrical torque";
Modelica.SIunits.Angle gamma =
p*(flange_a.phi - support.phi)

```

```

"Electr. angle betw. rotor and stator";
SIunits.Angle gammas =
  port_sp.reference.gamma
  "Angle in stator reference frame";
SIunits.Angle gammar =
  port_rp.reference.gamma
  "Angle in rotor reference frame";
equation
...
// Local balance of magneto motive force
// w.r.t. rotor fixed reference frame
(pi/2.0)*(V_mrr.re + V_msr.re) =
  Phi_rr.re*R_m.d;
(pi/2.0)*(V_mrr.im + V_msr.im) =
  Phi_rr.im*R_m.q;
// Torque
tauElectrical = -(pi*p/2.0)*
  (Phi_s.im*V_ms.re - Phi_s.re*V_ms.im);
flange_a.tau = -tauElectrical;
support.tau = tauElectrical;
// Stator may be potential root
Connections.potentialRoot
  (port_sp.reference);
Connections.branch(
  port_sp.reference, port_sn.reference);
port_sp.reference.gamma =
  port_sn.reference.gamma;
Connections.branch(
  port_rp.reference, port_rn.reference);
port_rp.reference.gamma =
  port_rn.reference.gamma;
Connections.branch(
  port_sp.reference, port_rp.reference);
gammas = gammar + gamma;
if Connections.isRoot(port_sp.reference)
then
  gammar=0;
end if;
end RotorSaliencyAirGap;

```

The references of the stator and rotor ports are linked through breakable connections. In case of generator operation without mains connection the stator will have to be treated as root.

#### 4.4 Damper Cages

For operating mains supplied synchronous machines it is required to enable the optional damper cage. Otherwise the induced rotor voltage for electrical and PM excited synchronous machines cannot perform the expected slow motion rotation. Since the load angle cannot change abruptly due to rotor inertia, the damper cage is required for operating quasi static synchronous machines with mains supply. The operating behavior of the quasi static machine model is yet different since all transient electrical effects are not taken into account.

The optional damper cages of the synchronous machines are equivalent two axis cages with different resistances and inductances in the rotor  $d$  (direct) and  $q$  (quadrature) axis. The squirrel cage of the induction machine model, however, has the same number of phases as the stator. Same stator and rotor phase numbers are chosen since it is usual in engineering approaches to model the squirrel cage equivalent to the topology of the stator winding.

#### 4.5 Example Machine Model

Figure 3 shows the permanent magnet synchronous machine model included in the quasi static fundamental wave library. The quasi static electrical domain is colored in light blue, the quasi static fundamental wave domain is light orange. The rotational and the thermal domain are colored in black and red, respectively.

The stray load loss model is directly part of the stator electric circuit. The voltage drop of this model is equal to zero, and loss is considered as torque times angular frequency. The stray load loss is also dissipated through the thermal connector.

The stator winding represents the electro magnetic coupling of the stator winding with magnetic field. The winding model is depicted in Fig. 4. This model consists of symmetrical winding phase resistors, an ideal electromagnetic coupling, a stray field reluctance in the magnetic domain and a loss model representing solely eddy current core loss. Thermal connectors for copper loss and eddy current loss are provided. Winding resistances are modeled temperature dependent, eddy current loss is modeled independent of temperature.

The air gap model couples the stator and rotor magnetic parts of the model. The electromagnetic torque is transmitted to the stator and rotor, respectively. Stator and rotor torques have the same absolute value but different signs. The magnetic saliency of the rotor is considered by different stator inductances with respect to the  $d$  (direct) and  $q$  (quadrature) axis. In the actual implementation the total inductances of the two axes are associated to one reluctance with respect to the  $d$  and  $q$  axis, respectively, only. Partial reluctances of the stator and rotor teeth, slots, yokes, etc. are currently not separated.

In the rotor magnetic circuit an optional damper cage is included. The damper cage model is a two axis model with cage resistances and stray inductances assigned to the two axis – this is the usual parametrization that electrical engineers are using. The damper cage also provides a thermal connector to exchange heat and temperature with an optional thermal model.

The permanent magnet model is currently not considering temperature dependent magnet properties. The total inductance of the machine already includes the reluctance of the permanent magnet. The PM model is thus a constant source of magnetic potential difference, rotated into the rotor fixed reference frame such that the complex magnetic potential difference phasor is aligned with the  $d$  axis. The mechanical flanges of the PM model are required to model the PM loss as equivalent mechanical loss torque times angular velocity.

The inertias, the friction loss model and the heat port and ambient models are the same for the quasi static and the transient fundamental wave machine models.

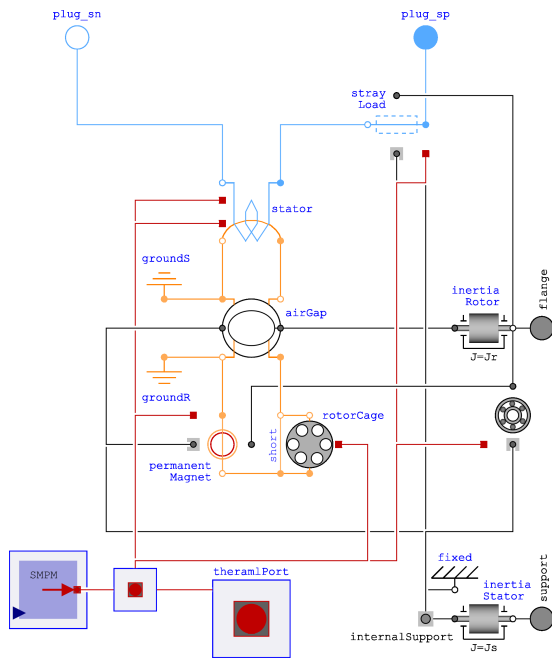


Figure 3: Quasi static model of a multi phase permanent magnet synchronous machine with permanent magnet

## 5 Library Compatibility

### 5.1 Parameter Compatibility

The new quasi static fundamental wave library is almost fully parameter compatible with the transient fundamental wave library. The only incompatibility are the stator zero inductances and the rotor zero inductance of the induction machine with wound slip ring rotor. These zero inductances are not implemented in quasi static machine models. In the transient machine models the default values of the zero inductances are the stray inductances. If the user does not propagate an actual parameter for the zero inductance of a transient machine model, the quasi static and transient machine models can be exchanged without parameter inconsistency.

In the current version of the quasi static fundamental wave library the loss models are fully compatible with the transient machine models. This may change in future versions where quasi static hysteresis loss may be included – which are difficult to implement for transient magnetic fields. Stator core loss, friction loss, stray load loss, permanent magnet loss and brush loss are implemented with compatible parameters and equal static state behavior.

### 5.2 Number of Phases

Parameter compatibility also includes the number of phases. In the quasi static fundamental wave library multi phase machines with phase numbers greater than or equal to three are supported.

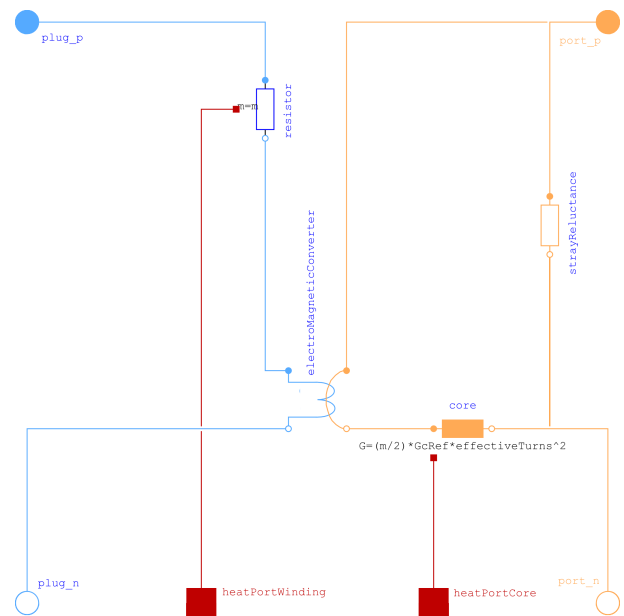


Figure 4: Winding model including winding resistors, electromagnetic coupling, stray reluctances and eddy current loss

### 5.3 Connector Compatibility

The electrical AC multi phase connectors of the transient and quasi static machine model are not compatible due to the different connector designs and properties. The analog DC connector of the synchronous machine with electrical excitation is yet connector compatible. The thermal connectors and the rotational connectors of the shaft and the optional housing, respectively, are connector compatible with the transient machine models.

## 6 Examples

In the following examples of transient and quasi static fundamental wave electric machines are compared. Each of the published examples is also available at the sub package `Examples.BasicMachines`.

### 6.1 Induction Machine with Squirrel Cage Rotor

Starting an induction machine with squirrel cage rotor direct on line shows a transiently higher starting and a lower break down torque than the quasi static machine; see Fig. 5 and 6. The simulation result is not plotted as a function of time, but as parametric plot of torques versus speed. Both machines are loaded with an idealized mechanical load, modeled as quadratic speed dependent torque. The example is available at `Examples.BasicMachines.AIMC_DOL`.

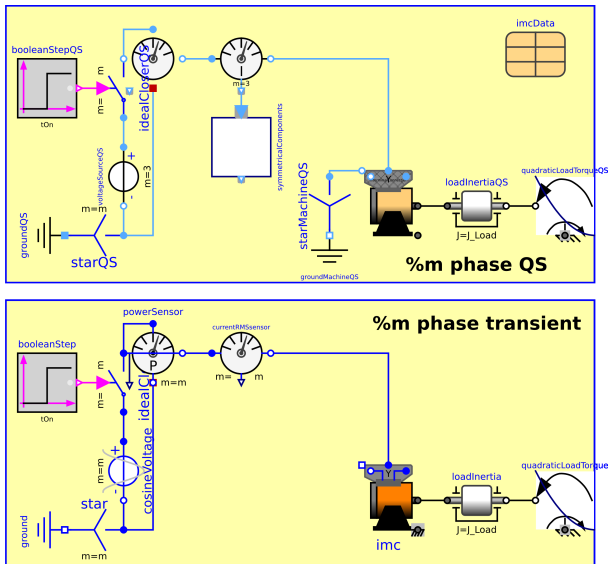


Figure 5: Modelica model of both transient and quasi static squirrel cage induction machine starting directly on line (DOL)

## 6.2 Electrical Excited Synchronous Generator

In example `Examples.BasicMachines.SMEE_Generator` a transient and quasi static synchronous generator with electrical excitation are compared for a very slow load change. The shafts of both generator are spinning with a rotational speed slightly different than synchronous speed, see Fig. 7. Therefore, the rotor is moved over one full electrical revolution – relative to the magnetic field. In the actual Modelica example the total real time of the experiment is equal to 30s. Such an experiment can also be performed in the lab for determining the full torque characteristic as function of the load angle. In Fig. 8 the torques of both machines are displayed versus angle  $\gamma_r$  obtained by the quasi static machine model. Angle  $\gamma_r$  and the load angle  $\vartheta$  are related by

$$\vartheta = \gamma_r - 90^\circ.$$

The torques of both machines are identical due to small relative speed between rotor and magnetic field. The wave form the torque is a sine wave superimposed with a sine wave of half the period due to the rotor saliency of this machine.

## 7 Possible Applications

### 7.1 Long Periods of Simulation Time

The new quasi static fundamental wave models may be used for all time domain simulations covering a large time span. Neglecting electrical transient effects makes the simulation models fast and robust. In particular, large periods of simulation time being in the range of minutes, hours and days, require fast simulation models. In typical simulation applications it is important to model power and energy balances accurately. High accuracy of the actual efficiency is

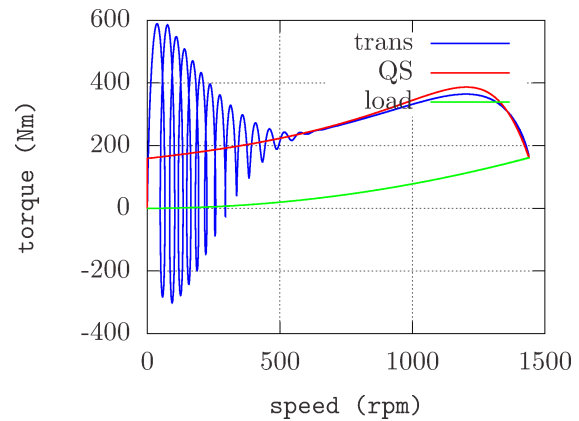


Figure 6: Torque versus speed for an induction machine with squirrel cage rotor, starting direct on line; comparing transient (trans) and quasi static (QS) machine model; load torque is quadratic speed dependent on speed

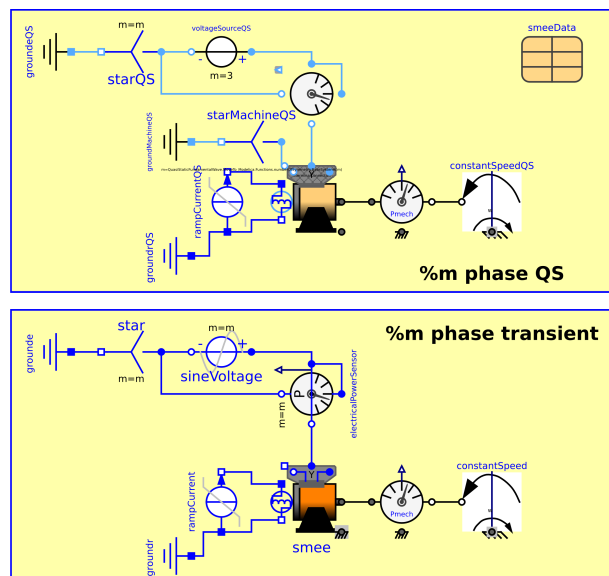


Figure 7: Modelica models of synchronous generators with electrical excitation

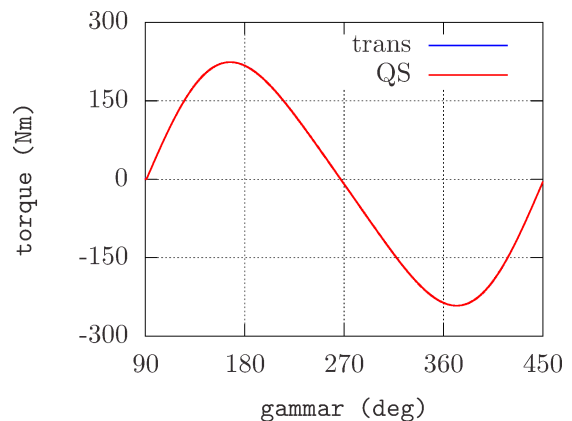


Figure 8: Torques of the transient and quasi static generator versus angle  $\gamma_r$

enabled by comprehensive loss models included in the machine models of the quasi static fundamental wave library. Typical applications are:

- Drive cycles of full or hybrid electric vehicles
- Drive cycles of subway trains, tramways, and railway trains
- Modeling of mechanical and electrical power and energy balances in industrial processes
- Robot drives for industrial use
- Auxiliary drives

## 7.2 Modularity of Simulation Concepts

Particularly for the simulation of electric machines including power electronics (converter) and control quasi static drive models are an interesting option. Quasi static and transient electric machine models are different levels of abstraction. The two different machine models can be combined with either a quasi static and transient converter models. Both the quasi static and the transient machine and converter can be operated by one control algorithm [5–10]. This way, the level of modularity of electric drives can be increased by means of the new quasi static fundamental wave machine models. The increased modularity reduces maintenance effort for drives libraries and reduces the development time of new or alternative control algorithms, in particular with respect to multi phase electric drives with phase numbers greater than three.

## 8 Didactic Aspects

In the quasi static and the transient fundamental wave library a strictly object oriented modeling approach was pursued. This includes the physical domains electrical, magnetic, rotational and thermal. Due to this approach the models of windings, the air gap and the permanent magnet, etc., can be clearly separated as different objects. In future versions it is even possible to separate the magnetic reluctances of teeth, slots, yokes and magnets.

Quasi static machine models are also of particular interest for high schools and universities to make virtual experiments and to demonstrate the physical behavior of electric machines. The different behavior and quality of quasi static and transient machine models can be discussed and elaborated in classes.

## 9 Conclusions

The new quasi static fundamental wave library for modeling induction and synchronous electric machines is presented. This package shall be included in the next version of the MSL. The provided machine models allow phase number equal to or greater than three. Fully symmetrical windings

and supply voltages and currents are required to fulfill quasi static modeling assumptions.

Additional modeling assumptions and limitations of the library are presented. The compatibility of the quasi static with the transient fundamental wave machine library is discussed. Simulation examples of quasi static and transient machine models are compared.

The new package opens a new field of applications for mobility and industry applications since the quasi static machine models have a very high simulation performance. Possible application examples are presented and didactic aspects of the library are discussed.

## References

- [1] C. Kral and A. Haumer, “The new fundamentalwave library for modeling rotating electrical three phase machines,” *8th International Modelica Conference*, 2011.
- [2] A. Haumer, C. Kral, J. V. Gragger, and H. Kapeller, “Quasi-stationary modeling and simulation of electrical circuits using complex phasors,” *International Modelica Conference, 6th, Bielefeld, Germany*, pp. 229–236, 2008.
- [3] N. Rabe, “An approach for modelling quasi-stationary magnetic circuits,” *9th International Modelica Conference*, 2012.
- [4] C. Kral, A. Haumer, and S. B. Lee, “A practical thermal model for the estimation of permanent magnet and stator winding temperatures,” *IEEE Transactions on Power Electronics*, vol. 29, no. 1, pp. 455–464, 2014.
- [5] A. Iqbal, E. Levi, M. Jones, and S. Vukosavic, “Generalised sinusoidal PWM with harmonic injection for multi-phase VSIs,” *PESC '06. 37th IEEE Power Electronics Specialists Conference, 2006*, 2006.
- [6] M. J. Duran, F. Barrero, and S. Toral, “Multi-phase space vector pulse width modulation: Applications and strategies.,” *Inernational Conference on Renewable Energies and Power Quality, ICREPQ 2007*, 2007.
- [7] D. Dujic, E. Levi, M. Jones, G. Grandi, G. Serra, and A. Tani, “Continuous PWM techniques for sinusoidal voltage generation with seven-phase voltage source inverters,” *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*, pp. 47–52, 2007.
- [8] S. Halasz, “PWM strategies of multi-phase inverters,” *IECON 2008. 34th Annual Conference of IEEE Industrial Electronics*, 2008.
- [9] D. Dujic, M. Jones, and E. Levi, “Generalised space vector PWM for sinusoidal output voltage generation with multiphase voltage source inverters,” *International Journal of Industrial Electronics and Drives*, vol. 1, no. 1, 2009.



- [10] M. Mengoni, *Modulation Techniques for Multi-Phase Converters and Control Strategies for Multi-Phase Electric Drives*. PhD thesis, University of Bologna, 2010.



# The New EDrives Library: A Modular Tool for Engineering of Electric Drives

Anton Haumer    Christian Kral  
Technical Consulting    Electric Machines, Drives and Systems  
3423 St.Andrä-Wördern, Austria    1060 Vienna, Austria  
anton.haumer@edrives.eu    christian.kral@edrives.eu  
www.edrives.eu

## Abstract

Simulation is an indispensable tool for the engineering of systems containing electric drives. Depending on the design phase and the engineering task different levels of modeling details are required: proof of concept, investigation of energy and power consumption, design of control, etc. The new EDrives library provides three levels of abstraction for inverters: quasi static (neglecting electrical transients), averaging (neglecting switching effects) and switching – for serving different demands. The inverters can feed the machine models of the Modelica Standard Library: `Modelica.Magnetic.FundamentalWave` and the new `Modelica.Magnetic.QuasiStatic.FundamentalWave`. The EDrives library copes with arbitrary phase numbers and can be easily extended to develop new control algorithms. In this publication the structure of the library and the implemented control principles are presented. Furthermore, examples comparing the three different levels of abstraction are included.

*Keywords: Electric machines and drives, power electronics, control, multi phase, thermal behavior, switching, quasi static, transient*

## 1 Introduction

Engineering of systems containing electric drives is essentially supported by simulation, allowing verification of the overall concept with rapid prototyping against requirements even in early design phases as shown in [1]. Different engineering tasks require different level of modeling abstraction: An early proof of concept can be done without knowledge of details like the control or switching algorithm of the inverter. Determination of energy and power consumption during driving cycles requires the accurate consideration of losses and the interaction of the control with the whole system. Yet, in such cases the switching effects can often be neglected. For more sophisticated investigations of, e.g., the torque ripple, a more detailed model is needed taking switching effects into account. Therefore a modular concept supporting all tasks is indispensable. The new EDrives

library provides inverter models with three levels of abstraction:

- Quasi static inverters connected to quasi static machine models; electrical transients are neglected, but losses are taken into account. The models have highest performance and robustness, making them suitable for long term simulations.
- Averaging inverters connected to transient machine models; the inverter models do not consider switching effects. Interaction of control with the whole system can be tested with high performance. Losses are taken into account, power conversion of the power electronics is based on power balance.
- Switching inverters provide the highest level of abstraction, resulting in lower simulation performance (i.e. longer execution time). On this level of abstraction switching effects and new pulse width modulation algorithms can be investigated.

The EDrives library supports arbitrary phase numbers  $m$  and is thus fully compatible with the `Modelica.Magnetic.FundamentalWave` and the new `Modelica.Magnetic.QuasiStatic.FundamentalWave` library. Yet the number of phases,  $m$ , being equal to  $2^n$  with integer  $n$  are currently not supported. The reasons are discussed in [2] and [3]. Segregating the inverter and the machine models allows the modular exchange with more advanced models of machines, power electronics and control. For example, an electric machine model can be coupled with a thermal model and cooling circuit to study the thermal behavior of the machine and system over a whole driving cycle.

The parameters of the entire drive are compiled in a drive parameter record. The drive parameter record contains a record of the electric machine, the power electronics and the control settings. This modular record structure enables the user to handle and exchange parameter sets for different machines and drives in a very convenient way.

In the actual version of the EDrives library control algorithms are provided for permanent magnet (PM) synchronous machines, solely. Control algorithms for in-

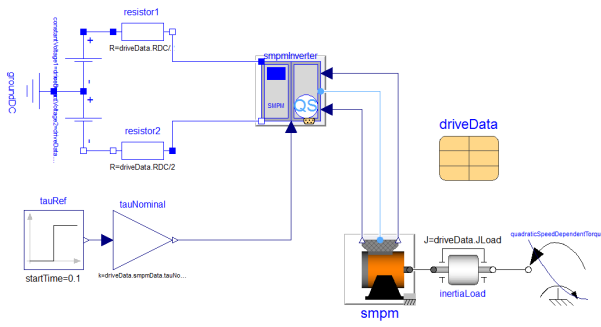


Figure 1: Permanent magnet synchronous machine with inverter

duction machines, electrical excited synchronous and synchronous reluctance machines as well as mains converters feeding the DC intermediate circuit will be implemented in upcoming versions. Therefore, all examples and models presented in this publication refer to permanent magnet synchronous machines.

## 2 Drive Concepts

Each **machine** model provided in the EDrives library consist of a machine model from the MSL including sensors for temperatures and the rotor flange angular position. The interface connectors are electrical, mechanical, thermal and the sensor outputs. These sensor outputs are connected with the inputs of the **inverter** model. The inverter consists of the power electronics, i.e., a **DC/AC converter**, sensors for electrical quantities and machine **control** and pulse width modulation (PWM) if required. The electrical AC output of the inverter is connected with a machine model and the DC input is connected with an external power supply. The type of DC power supply very much depends on the application. The DC power supply may either be a battery or fuel cell or an AC to DC converter supplied by the mains. An example of a PM synchronous machine drive modeled with the EDrives library is shown in Fig. 1. A detailed description of the main drive components is presented in sections 3–6.

## 3 Machines and Sensors

The EDrives library will provide wrapper models for each type of machine to encapsulate the machine model including sensors, as shown in Fig. 2 for a PM synchronous machine. These wrapper models contain:

- an instance of a quasi static or transient machine model of the MSL
- a terminal box, providing the desired terminal connection (star or polygon)
- a rotor angle sensor
- temperature sensors for making stator winding and rotor (PM) temperatures accessible

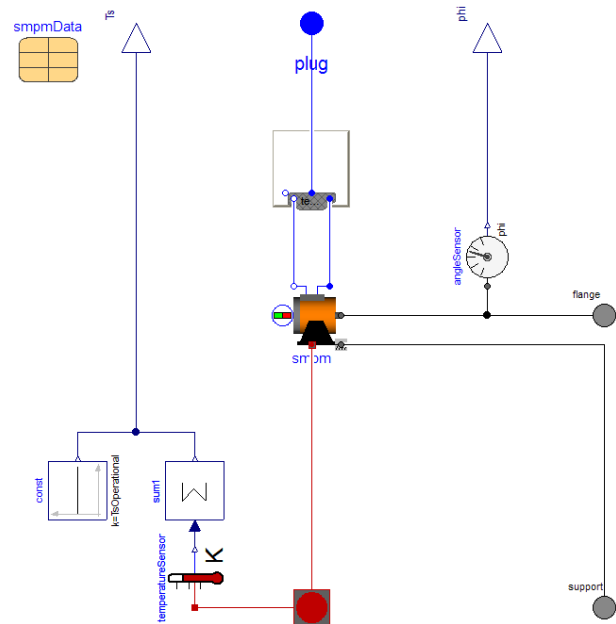


Figure 2: Wrapper model for the transient machine

- the machine parameter record

The machine parameter record allows to propagate all machine parameters with one record. This is achieved by the wrapper model propagating the elements of the parameter record to the machine model. The concept of the wrapper model enables the exchange of a quasi static by a transient machine model in an application and vice versa. The parameter records and all connectors except for the electrical connector are compatible in each wrapper model. This way only the electrical machine supply has to be re-connected. The mechanical flanges, the thermal connector as well as the temperature sensors and angle sensor outputs are the same for all wrapper models of permanent magnet synchronous machines.

Both the quasi static and the transient machine models take losses into account: temperature dependent copper loss, (eddy current) core loss, friction loss, stray load loss and PM loss – see [4, 5]. All these losses are dissipated either to an internal thermal ambient with fixed operational temperature, or exchanged with an external thermal model through the thermal port. An external thermal model allows to simulate the actual temperatures over a driving cycle as discussed, e.g., in [6, 7].

## 4 Inverters and Parametrization

Each inverter model extends from the same base class since they all have the same connectors except for the electrical plug that has to be connected to the machine. However, each inverter model uses the same drive parameter record. Thus, it is possible to replace one inverter model by another in an experiment with low effort. However, the level of abstraction of the machine model has to match level of abstraction of the inverter model: a quasi static inverter can feed a quasi



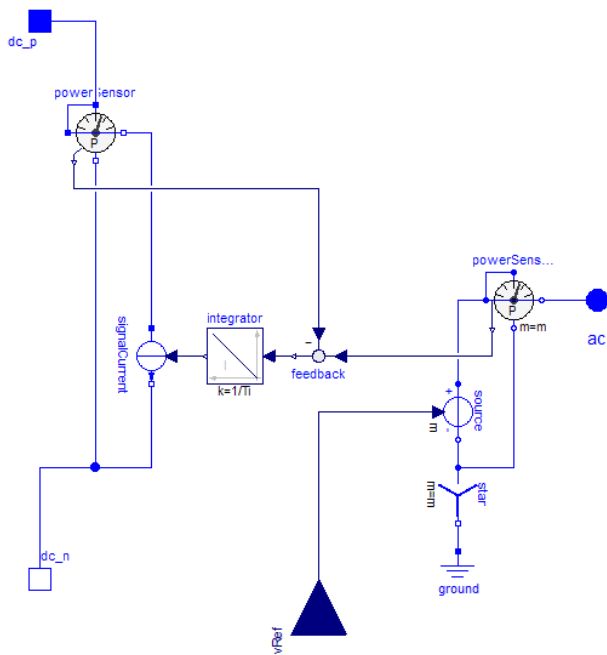


Figure 4: Averaging converter model

eters of the drive configuration. The controller parameters are calculated from the machine parameters and propagated to the controller models. The user can tune the pre-configured control parameters by means of tuning parameters.

## 5 DC / AC Converter

Three different levels of abstraction for DC to AC converters are provided in the EDrives library.

- First, the quasi static converter is modeled as an ideal quasi static voltage source. Power balance between DC input and AC output is complied. This means the efficiency of the quasi static converter is 100%.
- Second, in the averaging converter model (Fig. 4) the space phasor voltages are transformed into transient phase voltages. These voltages are fed to an ideal voltage source. Considering power balance between DC input and AC output side, the converter efficiency is 100%.
- Third, the switching converter consists of  $m$  legs. Each of them consists of two electronic switches and two anti parallel free-wheeling diodes. Switch and diode models are taken from the Modelica Standard Library (`IdealGTOThyristor` and `IdealDiode` from package `Modelica.Electrical.Analog.Ideal`). The inputs of the switching converter are boolean switching signals (`on|off`) for each leg.

## 6 Control

Machine control is based on FOC (field oriented control) and consists of the following components:

- optional speed controller
- torque flux controller
- current controller

In each controller model, a space vector representation of voltages and currents in the rotor fixed reference frame is used. These phasors are also distributed to the machine bus. The input for the torque-flux-controller is the commanded torque which can be provided by the user or an optional speed controller.

### 6.1 Optional Speed Controller

The external speed controller is implemented as limited PI controller with anti windup. It feeds the commanded torque via signal connector to the inverter and has to be connected to the inverter via machine bus. The parametrization is defined by a record `speedControllerData` which is a sub-record of the drive parameter record. The record `speedControllerData` calculates the parameters for the PI controller based on the symmetrical optimum method. The subscripts  $iq$  refer to the settings of the current controller with respect to the  $q$  axis [8]:

$$k_p = k_{Tune} \frac{(J_{Machine} + J_{Load}) \omega_{Nominal}}{\tau_{Nominal}} \frac{2k_{Tune,iq}}{T_i}$$

$$T_i = T_{i,iq} \frac{4}{k_{Tune,iq}}$$

### 6.2 Torque Flux Controller

The torque-flux-controller shown in Fig. 5 limits the commanded torque with the maximum speed dependent torque. In the field weakening range torque is limited proportional to one over speed. In the base speed range the maximum torque is constant. The actual machine torque, magnetic flux and stator voltage are determined by a controller internal machine model (section 6.4). The input quantities of the controller internal machine model are the commanded current space vector and the rotor angular position. In a real drive, torque, magnetic flux and stator voltage are either determined by an observer or an on-line model which is implemented in the control software.

The deviation between the commanded and the modeled torque is the input for a fast integrator controlling the  $q$  current component.

The  $d$  current component is determined by the flux controller which implements a simple flux control algorithm. It can be replaced by a more sophisticated flux controller with low effort. First, the actual voltage, calculated by the controller internal machine model is compared with the maximum admissible voltage. This is the minimum of the maximum admissible machine voltage defined by a parameter

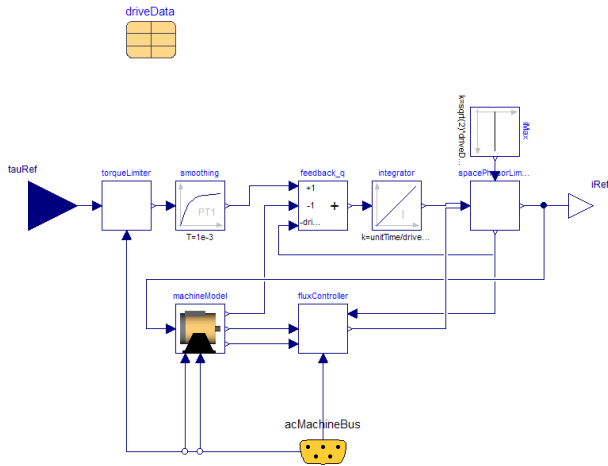


Figure 5: Torque flux controller

and the maximum obtainable voltage according to the actual DC voltage. Second, the actual magnetic stator flux of the machine is compared with the nominal magnetic flux. Nominal flux is calculated by an auxiliary quasi static machine model to determine the nominal operating conditions defined by the machine parameter record. If the actual voltage exceeds the maximum admissible voltage, field weakening occurs. When the actual magnetic flux exceeds the nominal magnetic flux, field weakening ends, entering the base speed range where magnetic flux is kept constant.

In the constant flux region, the  $d$  current component is controlled by a fast integrator to achieve constant magnetic flux. In the field weakening region, the  $d$  current component is determined by the same integrator to keep the voltage limited.

The outputs of the torque-flux-controller are the commanded  $d$  and  $q$  current component of the current space phasor in the rotor fixed reference frame. These components are limited to the maximum admissible machine current, which is defined in the drive parameter record. The limited current space phasor is equal to the commanded current space phasor which is forwarded to the current controller. The commanded current space phasor is also fed to the controller internal machine model.

### 6.3 Current Controller

The current controller for the transient inverters (averaging and switching) utilizes two limited PI controllers with anti windup to determine the stator voltages  $V_d$  and  $V_q$ , respectively. The stator voltage space phasor is determined such way that the commanded currents,  $\hat{I}_d$  and  $\hat{I}_q$ , and the actual currents,  $I_d$  and  $I_q$ , coincide.

The voltage equations of the PM synchronous machine are

$$V_d = R_s \hat{I}_d - \omega L_q \hat{I}_q + L_d \frac{d\hat{I}_d}{dt},$$

$$V_q = R_s \hat{I}_q + \omega L_d \hat{I}_d + L_q \frac{d\hat{I}_q}{dt} + V_{PM},$$

where  $\omega$  is the angular rotor speed. Parameter  $R_s$  represents the stator resistance at rated operating temperature, and  $L_d$  and  $L_q$  are the total inductances of the  $d$  and  $q$  axis. The controller internal machine model determines the voltage drops  $R_s \hat{I}_d + \omega L_q \hat{I}_q$  and  $R_s \hat{I}_q + \omega L_d \hat{I}_d + V_{PM}$ , which are added as feed forward signal to the PI controller outputs. This way the PI controllers solely controls the transient current deviations  $\frac{d\hat{I}_d}{dt}$  and  $\frac{d\hat{I}_q}{dt}$ ; in literature this technique often is called “decoupling”. In the field weakening region the desired voltage space phasor is limited to the maximum admissible voltage (section 6.2).

Since the machine transfer function represents a first order delay, the parametrization of the PI controllers is based on the compensation method [8].

$$k_{p,id} = k_{Tune,id} R_s$$

$$T_{i,id} = \frac{L_d}{R_s}$$

$$k_{p,iq} = k_{Tune,iq} R_s$$

$$T_{i,iq} = \frac{L_q}{R_s}$$

These controller parameters are calculated by the drive parameter record.

The quasi static current controller is simpler: The voltages to match the actual and commanded  $d$  and  $q$  currents are calculated by the controller internal machine model. The complex voltage is limited to the maximum admissible voltage and fed to the output.

### 6.4 Controller Internal Machine Model

The controller internal machine model shown in Fig. 6 is used to calculate the stator voltage phasor, the magnetic flux and torque related to the the commanded stator current space phasor and rotor position. The actual stator winding temperature, however, is taken into account by the controller internal machine model. This machine model is implemented as the instance of a quasi static machine model as described in [3].

Using a quasi static machine model to implement the controller internal machine model is much more structured and modular than an model just implementing text book equations describing the machine’s behavior. Additionally, all improvements of the machine model from the MSL can be utilized. Machine losses and temperature effects are fully taken into account, resulting in a flexible and adaptive machine model.

### 6.5 Space Vector PWM

SVPWM (space vector pulse width modulation) is considered in literature as state of the art PWM, as discussed, e.g., in [9–13]. The PWM block included in the actual version

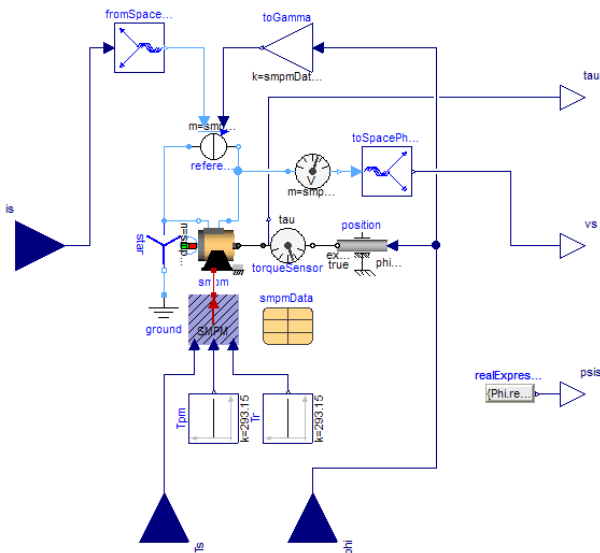


Figure 6: Controller internal machine model

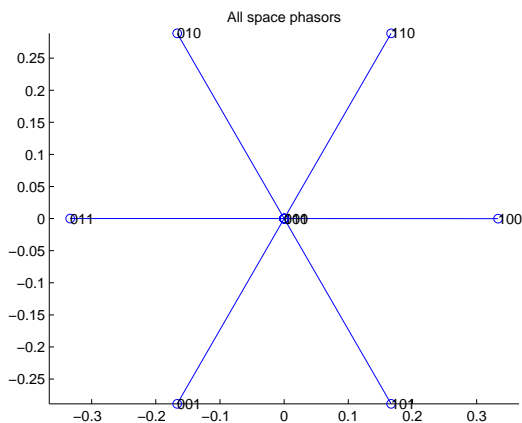


Figure 7: SVPWM switching states in d-q-voltage plane [p.u.] for  $m = 3$  phases

of the EDrives library currently implements a SVPWM algorithm for  $m = 3$  phases. An extension to arbitrary number of phases is under development. The algorithm samples the desired voltage space phasor and the actual DC voltage according to the switching frequency. The possible space phasor locations according to the switching states shown in Fig. 7 are calculated by a function. First, the sector is determined where the commanded voltage space phasor is located. This voltage space phasor can be composed of the two neighbored switching states and either of the two zero switching states. The boolean switching states are fed to the output, to control the electronics switches of the converter (section 5).

## 7 Examples

The first example shown in Fig. 1 simulates a quasi static synchronous machine with permanent magnets, fed by a

quasi static inverter. For the given drive configuration three different levels of abstraction will be compared. First, the original simulation based on the quasi static machine and inverter is performed. Second, the permanent magnet synchronous machine model `smpm` is redeclared as transient machine. Additionally, the inverter model `smpmInverter` feeding the permanent magnet synchronous machine has to be redeclared as averaging inverter. Then an additional simulation run is performed. Third, the averaging inverter is redeclared by a switching inverter.

The machine parameters of the simulation example are derived from the default MSL machine parameters summarized in Tab. 1. No damper cage is used; copper losses are caused by heat dissipation of the stator resistors. All other losses are neglected in the preformed example. The machine is mechanically loaded by an inertia and a quadratic speed dependent torque which reaches nominal torque at nominal speed, representing a fan or a pump. The inverter is connected to a DC voltage source with resistors, simulating a battery. At  $t = 0.1$  s a step from zero to nominal torque is applied to the reference torque input. After accelerating, the drive reaches an equilibrium with the given load characteristic.

The following figures compare the results of

- a transient synchronous machine fed by a switching inverter,
- a transient synchronous machine fed by an averaging inverter and
- a quasi static synchronous machine fed by a quasi static inverter.

In Fig. 8 the actual torques of the three cases follow the reference torque step very well. As expected, the quasi static torque is nearly identical with the reference torque. The switching case shows a torque ripple according to the switching frequency (5 kHz). The speed trajectories in Fig. 9 show only small differences. The plots of the RMS stator currents in Fig. 10 reveal that the  $d$  current increases at  $t = 0$  s to set the nominal magnetic flux. At  $t = 0.1$  s the  $q$  current component is controlled to generate the reference torque. Again the quasi static case shows no transients, whereas the switching current is superimposed by a rest of current ripple left by filtering. The unfiltered phase currents at the end of the simulation are depicted in Fig. 11. A fast Fourier transform (FFT) of phase current of phase 1 is shown in Fig. 12. Note that the largest harmonics – apart from the fundamental wave – are located at multiples of the switching frequency (5 kHz). The filtered stator RMS voltages in Fig. 13 rise with speed since induced voltage increases. Yet the maximum voltage of 100 V is not exceeded. The second example shown in Fig. 14 is derived from the first example, but the number of phases is changed to  $m = 5$ . The desired torque step is replaced by a speed controller which input is a desired speed cycle, and the parametrization of the speed controller is taken from the record `speedControllerData`, embedded in the drive parameter record.



number of phases	$m$		arbitrary
number of pole pairs	$p$		2
rotor's moment of inertia	$J_r$	kg · m <sup>2</sup>	0.29
nominal frequency	$f_{s,Nominal}$	Hz	50
open circuit voltage at nominal speed	$V_{s,OpenCircuit}$	V	112.3
nominal voltage per phase	$V_{s,Nominal}$	V	100
nominal current per phase	$I_{s,Nominal}$	A	$100 \cdot \frac{m}{3}$
stator resistance per phase at 20°C	$R_s$	Ω	$0.03 \cdot \frac{m}{3}$
stator stray inductance per phase	$L_{s\sigma}$	H	$\frac{0.1}{2\pi f_{s,Nominal}} \cdot \frac{m}{3}$
stator main field inductance per phase, d axis	$L_{md}$	H	$\frac{0.3}{2\pi f_{s,Nominal}} \cdot \frac{m}{3}$
stator main field inductance per phase, q axis	$L_{mq}$	H	$\frac{0.3}{2\pi f_{s,Nominal}} \cdot \frac{m}{3}$

Table 1: Machine parameters

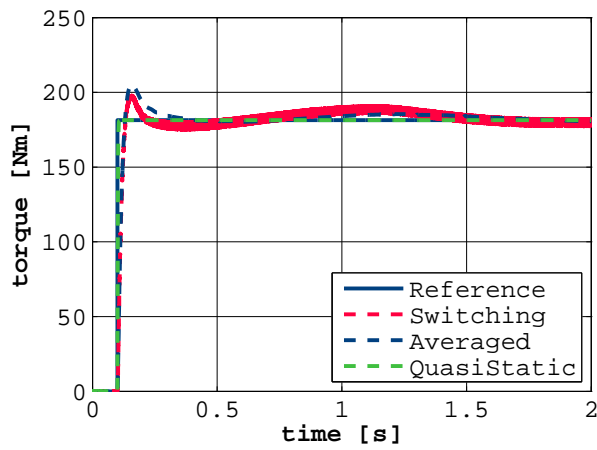


Figure 8: Example 1: simulation results of electrical torque

The simulation results are obtained with a transient machine model and an averaging inverter model. Fig. 15 shows the reference torque given by the speed controller and the machine electrical torque. The actual speed follows reference speed very well, as shown in Fig. 16. Fig. 17 shows the 5 phase currents during 1 second of the initial acceleration. This proves that the control system works well in both driving directions, both for motor and generator mode, and for phase numbers  $m > 3$ .

## 8 Outlook

It is planned to release the new EDives library as commercial tool-independent library. The modular concept of the library supports the following development steps:

- Enhancing the SVPWM algorithm to arbitrary numbers of phases  $m > 3$ .
- Extending both the switching converter model and the SVPWM algorithm to multilevel inverter configurations.
- Considering alternative flux control strategies.
- Implementing the control structures for induction machines with squirrel cage; power electronic converters

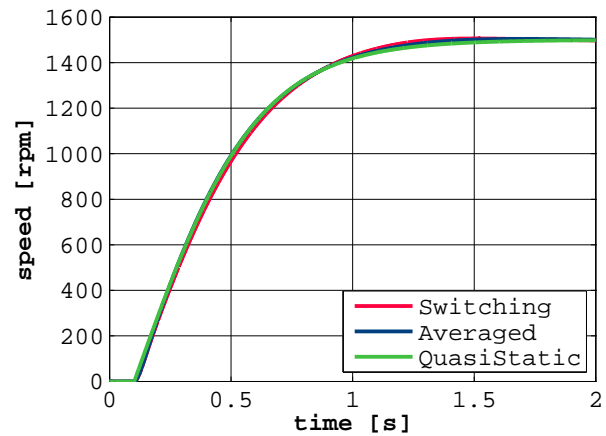


Figure 9: Example 1: simulation results of speed

(quasi static, averaging and switching) as well as the SVPWM algorithm can be re-used.

- Providing diode bridge converters as mains converters (quasi static, averaging and switching).
- Developing active mains converters with voltage-oriented control as described in [14]; power electronic converters (quasi static, averaging and switching) as well as the SVPWM algorithm can be re-used.
- In the current implementation, the switching power converter models using the `IdealGTOThyristor` and the `IdealDiode` model from package `Modelica.Electrical.Analog.Ideal` take conduction losses into account. Enhancements include implementation of switching losses for switching power converters, and both conduction and switching losses for averaging and quasi static power converters. One particular focus of development is a user friendly parametrization technique. Furthermore, thermal effects shall be considered, including a thermal port for coupling the electrical models with thermal inverter models and cooling circuits.
- Adaptive control techniques shall be considered, taking temperature dependencies and saturation effects into account.

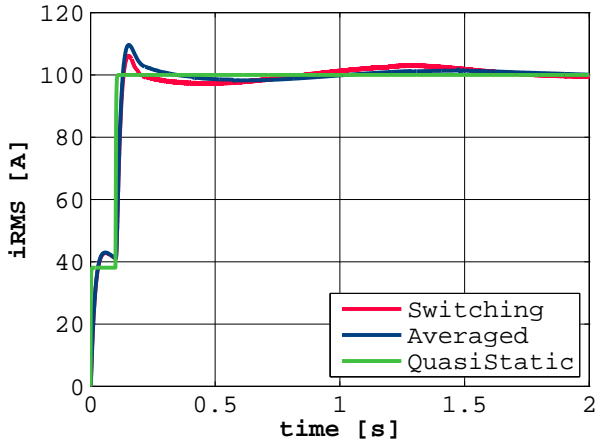


Figure 10: Example 1: simulation results of stator RMS current

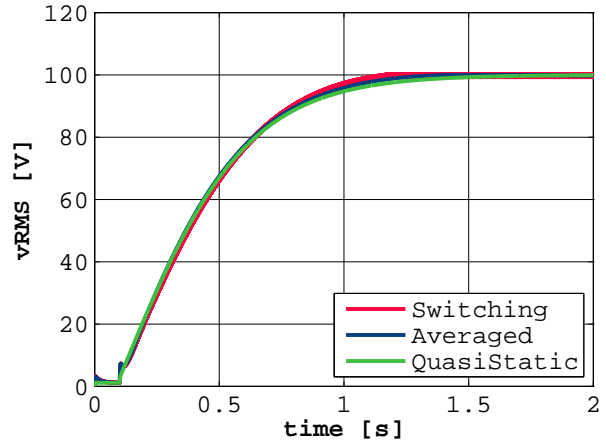


Figure 13: Example 1: simulation results of filtered stator RMS voltage

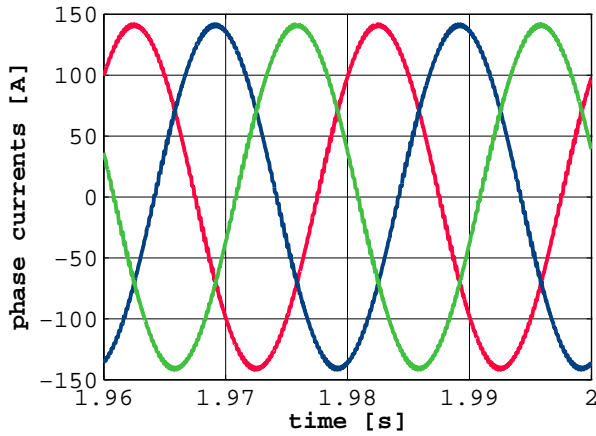


Figure 11: Example 1: simulation results of stator phase currents

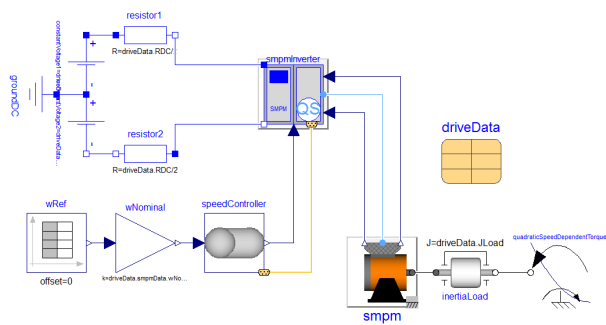


Figure 14: Example 2: speed controlled PM synchronous machine drive

## 9 Conclusions

The new EDrives library for modeling inverter drives utilizes the machine models from the MSL. The provided in-

verter models allow phase numbers equal or greater than 3, like the `Modelica.Magnetic.FundamentalWave` and the new `Modelica.Magnetic.QuasiStatic.FundamentalWave`. The control structures are based on FOC. The parametrization of the controllers is described. Simulation examples demonstrate the usage of matching inverter and machine models with respect to different levels of abstraction.

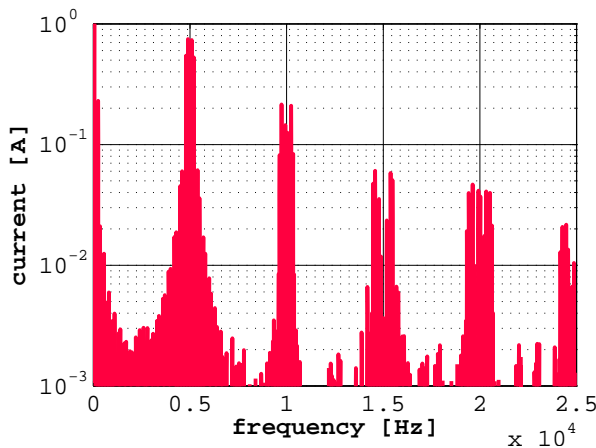


Figure 12: FFT of simulation of stator RMS current at equilibrium

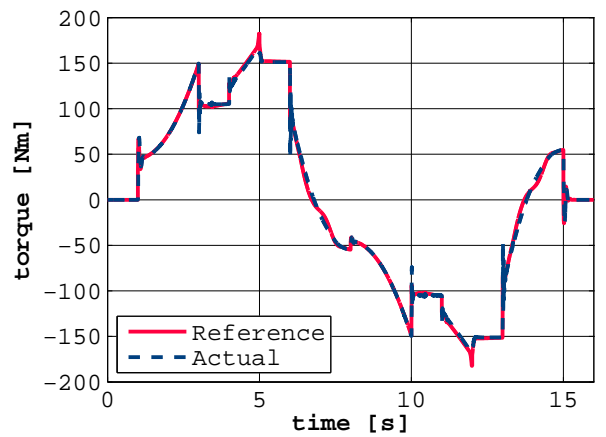


Figure 15: Example 2: simulation results of electrical torque

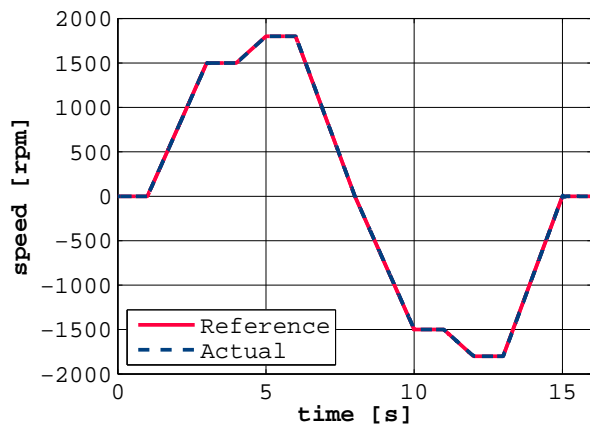


Figure 16: Example 2: simulation results of speed

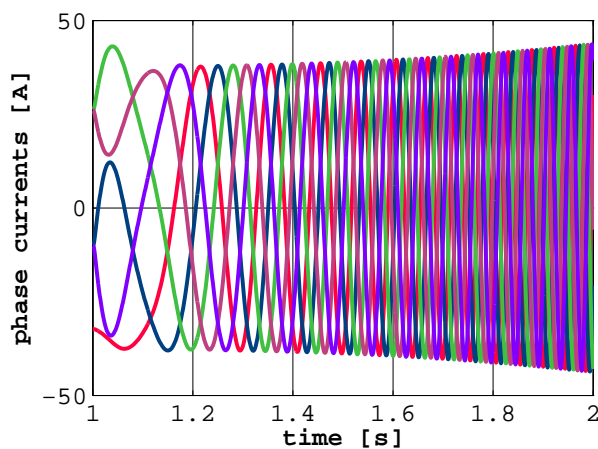


Figure 17: Example 2: simulation results of phase currents

The new EDrives library opens a new field of applications for mobility and industrial applications: Quasi static models with highest performance allow long term energy consumption simulations. Averaging models enable testing the interaction of the drive control with the whole system at high performance. The most detailed switching models allow the investigation of ripple effects and the development and customization of controller concepts and designs.

The package shall be released as a commercial library, the first version comprises quasi static, averaging and switching inverter models for synchronous machines. Enhancements and extensions are currently under development.

## References

- [1] A. Haumer, T. Bäuml, and C. Kral, "Multiphysical simulation improves engineering of electric drives," *7th EUROSIM Congress on Modelling and Simulation*, September 2010.
- [2] C. Kral, A. Haumer, and R. Wöhrnschimmel, "Extension of the fundamentalwave library towards multi phase electric machine models," *submitted for possible publication at Modelica Conference*, 2014.
- [3] C. Kral and A. Haumer, "New multi phase quasi static fundamental wave electric machine models for high performance simulations," *submitted for possible publication at Modelica Conference*, 2014.
- [4] A. Haumer, C. Kral, H. Kapeller, T. Bäuml, and J. V. Gragger, "The AdvancedMachines library: Loss models for electric machines," *Proceedings of the 7th Modelica Conference*, pp. 847–854, 2009.
- [5] C. Kral and A. Haumer, *Object Oriented Modeling of Rotating Electrical Machines*. INTECH, 2011.
- [6] C. Kral, A. Haumer, and M. Plainer, "Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library," *International Modelica Conference, 4th, Hamburg, Germany*, pp. 213–218, 2005.
- [7] A. Haumer, C. Kral, V. Vukovic, A. David, C. Hettfleisch, and A. Huzsvar, "A parametrization scheme for high performance thermal models of electric machines using Modelica," *MATHMOD VIENNA 2012*, 2012.
- [8] P.-I. H. Lutz and P. D. I. W. Wendt, *Taschenbuch der Regelungstechnik, 5. erweiterte Auflage*. Frankfurt am Main: Wissenschaftlicher Verlag Harri Deutsch, 2003.
- [9] M. J. Duran, F. Barrero, and S. Toral, "Multi-phase space vector pulse width modulation: Applications and strategies.," *Inernational Conference on Renewable Energies and Power Quality, ICREPQ 2007*, 2007.
- [10] S. Halasz, "PWM strategies of multi-phase inverters," *IECON 2008. 34th Annual Conference of IEEE Industrial Electronics*, 2008.
- [11] D. Dujic, M. Jones, and E. Levi, "Generalised space vector PWM for sinusoidal output voltage generation with multiphase voltage source inverters," *International Journal of Industrial Electronics and Drives*, vol. 1, no. 1, 2009.
- [12] M. Mengoni, *Modulation Techniques for Multi-Phase Converters and Control Strategies for Multi-Phase Electric Drives*. PhD thesis, University of Bologna, 2010.
- [13] D. G. Holmes and T. A. Lipo, *Pulse Width Modulation for Power Converters*. Wiley Interscience, 2003.
- [14] A. Haumer and C. Kral, "Modeling a mains connected PWM converter with voltage-oriented control," *Modelica Conference*, 2011.



# Modelica Models for Magnetic Hysteresis, Materials and Transformers

Johannes Ziske, Thomas Bödrich

Technische Universität Dresden, Institute of Electromechanical and Electronic Design  
01062 Dresden, Germany

Johannes.Ziske@tu-dresden.de

Thomas.Boedrich@tu-dresden.de

## Abstract

Within the Clean Sky project MoMoLib (Modelica Model Library Development for Media, Magnetic Systems and Wavelets) an extension for the Modelica.Magnetic.FluxTubes library has been developed. This extension mainly consists of new flux tubes elements for the consideration of the magnetic hysteresis in the transient simulation of electromagnetic networks, a materials library with hysteresis data of various magnetic materials and a new components package with models of one- and three-phase transformers, which also account for the magnetic hysteresis of the core. This paper briefly presents the implemented hysteresis models for the simulation of the static (ferromagnetic) and dynamic (eddy currents) hysteresis. It shows the accurate computation of the instantaneous hysteresis losses, which becomes increasingly important for the design of electromagnetic components with increasing requirements regarding energy efficiency and mass power densities. Furthermore, the new components of the library extension are introduced and the behavior of the implemented elements is verified and compared to measurements and steel sheet datasheets.

*Keywords: magnetic hysteresis; Tellinen; Preisach; Modelica.Magnetic.FluxTubes; iron losses;*

## 1 Introduction

The Modelica.Magnetic.FluxTubes library was originally developed at the Technische Universität Dresden and has been part of the Modelica Standard Library (MSL) [1] since 2009. The library is based on the well-known concept of magnetic flux tubes [2, 3] and allows modeling of magnetic fields with lumped networks. Due to the library elements for modeling of coils, non-linear core material, leakage flux and reluctance forces this library is well-suited for the rough design of electromagnetic components and devices, e.g. actuators, motors, transformers, or hold-

ing magnets. Important properties of magnetic components, e.g. saturation behavior, residual magnetism and especially the iron losses are influenced or even determined by the magnetic hysteresis behavior of the involved ferromagnetic materials. Particularly, the consideration of hysteresis losses gets increasingly important during the design process of magnetic components due to increasing requirements on loss power minimization and high mass power densities. Well-known examples of this engineering trend are e.g. the electromobility and more electric aircraft. So far, the ferromagnetic hysteresis is not yet considered in the FluxTubes library. Within a Clean Sky project this issue has been addressed and an extension for the FluxTubes library has been developed. It includes hysteresis elements for the consideration of both static and dynamic hysteresis of ferromagnetic materials. At the Modelica 2012 conference first Modelica hysteresis models have been presented [4]. Now the work on the extension is almost finished and the extended Modelic.Magnetic.FluxTubes library will soon be integrated into the MSL. Two different hysteresis models have been implemented for modeling of static ferromagnetic hysteresis. On the one hand the rather simple but efficient Tellinen model [5], which is characterized by low computational effort and high numerical stability, and on the other hand the more complex but also more accurate and widely accepted Preisach hysteresis model [6]. Dynamic hysteresis is computed by the product of the classical eddy current factor [5, 7], which considers the electrical conductivity and the thickness of magnetic steel sheets, and the time derivative of the magnetic flux density, which causes the formation of eddy currents. To configure the hysteresis models and to adapt them to specific materials the FluxTubes.Material package has also been extended with a package containing hysteresis properties of typical magnetic materials. This data is mainly based on in-house measurements according to DIN EN 60404-2 using a 25 cm Epstein frame. With a series of simulations the behavior of the developed hysteresis models has been verified and compared to published

data of a steel sheet manufacturer as well as to in-house measurements.

## 2 Theory

This section gives a short introduction to the two implemented static hysteresis models, the Tellinen and the Preisach model, and introduces the approach for consideration of eddy currents and hysteresis losses.

### 2.1 The Tellinen Hysteresis Model

The Tellinen hysteresis model [5] is a comparatively simple model for the description of ferromagnetic (static) hysteresis. Thus, it is easy to implement, numerically stable and fast. Even though the model does not have a magnetic memory, its accuracy is sufficient for a wide range of simulations.

To adapt the model to specific material properties (hysteresis shapes) only the hysteresis envelope curve, i.e. the rising  $R(H)$  and falling  $F(H)$  branches of the limiting hysteresis loop, must be provided (see Figure 1). Together with their corresponding slope functions  $r(H)$  and  $f(h)$ , which define the slope of  $R(H)$  and  $F(H)$  with respect to the magnetic field strength  $H$ , the actual slope  $dB/dt$  of the current operation point  $O(h, b)$  is given by equation (1).

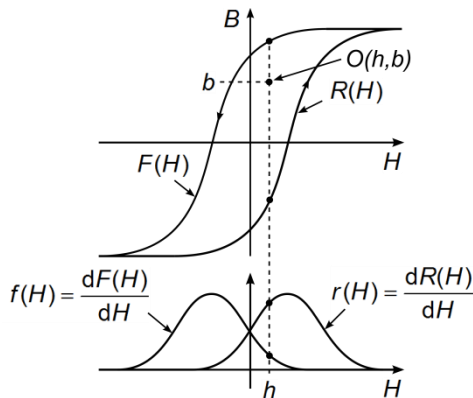


Figure 1: Rising  $R(H)$  and falling  $F(H)$  branch of the hysteresis envelope curve and their corresponding slope functions  $r(H)$  and  $f(H)$

$$\frac{dB}{dt} = \begin{cases} \frac{F(H) - B}{F(H) - R(H)} \cdot r(H) \cdot \frac{dH}{dt} & \text{for } \frac{dH}{dt} > 0 \\ \frac{B - R(H)}{F(H) - R(H)} \cdot f(H) \cdot \frac{dH}{dt} & \text{for } \frac{dH}{dt} < 0 \end{cases} \quad (1)$$

### 2.2 The Preisach Hysteresis Model

This section shows the implementation improvements compared to the prior state of the model presented at the 9<sup>th</sup> Modelica conference 2012 [4]. The Preisach model is described in more detail e.g. in [6, 8]. The behavior of the Preisach model results from a superposition of an infinite set of elementary hysteresis operators  $\gamma_{\alpha\beta}$  with the upper and lower switching limits of  $\alpha$  and  $\beta$ , respectively. The operators output equals +1 for a magnetic field strength  $h$  greater than  $\alpha$  and -1 for  $h$  less than  $\beta$ . For all  $h$  between  $\alpha$  and  $\beta$ , the output  $\gamma_{\alpha\beta}h(t)$  remains in the previous state (see equation (2) and Figure 2).

$$\gamma_{\alpha\beta}h(t) = \begin{cases} -1 & \text{for } h(t) \leq \beta \\ +1 & \text{for } h(t) \geq \alpha \\ \text{previous} & \text{else.} \end{cases} \quad (2)$$

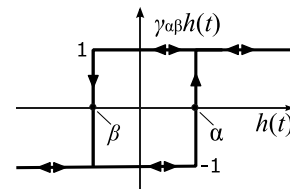


Figure 2: Elementary Preisach operator  $\gamma_{\alpha\beta}$  (hysteron).

With the restriction that  $\alpha$  is always greater than  $\beta$ ,  $\alpha$  and  $\beta$  span a right-triangular plane which is often referred to as Preisach plane (see Figure 3).

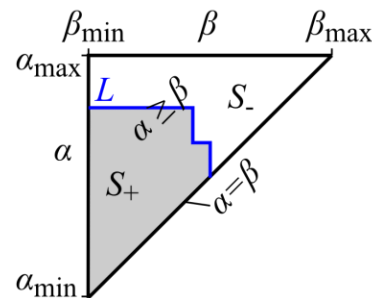


Figure 3: Preisach plane

For each point  $(\alpha, \beta)$  of this region exactly one elementary hysteresis operator  $\gamma_{\alpha\beta}$  exists with the switching limits of exactly  $\alpha$  and  $\beta$ . The Preisach distribution function  $P(\alpha, \beta)$  is also defined over this Preisach plane and gives a specific weight to each operator (see Figure 4).

The polarization  $j(t)$  of the model is then defined by the integral over all weighted operators outputs multiplied by the saturation polarization  $J_S$ :

$$j(t) = J_S \cdot \iint_{\alpha \geq \beta} P(\alpha, \beta) \cdot \gamma_{\alpha\beta} h(t) d\alpha d\beta. \quad (3)$$

If one splits the Preisach plane into two regions S+ and S-, in which all the operators are in the +1 and -1 state, respectively, equation (3) can be simplified to

$$j(t) = J_S \cdot \left( 2 \cdot \iint_{S_+(t)} P(\alpha, \beta) d\alpha d\beta - 1 \right). \quad (4)$$

It is evident from equation (4) that the Preisach distribution function defines the shape and the behavior of the hysteresis.

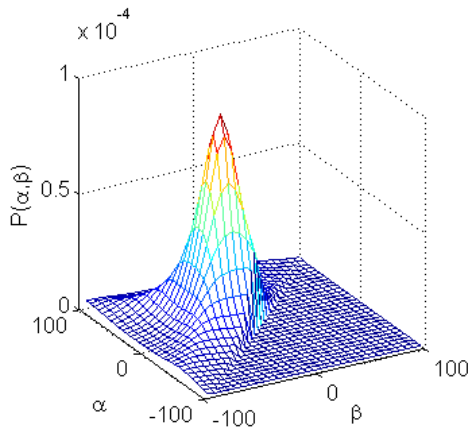


Figure 4: Exemplary Preisach distribution function  $P(\alpha, \beta)$  defined over the Preisach plane ( $\alpha \geq \beta$ ).

Normally, the double integral of  $P(\alpha, \beta)$  is not analytically defined. A numerical double integration at every time step of the simulation would be too computationally expensive. In the initial version of the Preisach flux tube element [4] that problem was solved by integrating the Preisach distribution function only once at the start of each simulation for each grid point of a fixed grid over the Preisach plane. The results had been stored in a two-dimensional array (CombiTable2D), which then was used during simulation for table lookup and interpolation to evaluate the integral. The improved version of the element uses now another approach instead, which again improves simulation speed and numerical stability, namely an analytical description of the Everett function [9]. This Everett function  $E_V(\alpha, \beta)$  is defined as the integral of  $P(\alpha, \beta)$  over the region R:

$$\begin{aligned} E_V(\alpha', \beta') &= \iint_R P(\alpha, \beta) d\alpha d\beta \\ &= \int_{\alpha=\beta'}^{\alpha'} \int_{\beta=\beta'}^{\alpha} P(\alpha, \beta) d\beta d\alpha. \end{aligned} \quad (5)$$

Figure 5 shows the region R with its integration limits  $\alpha'$  and  $\beta'$ .

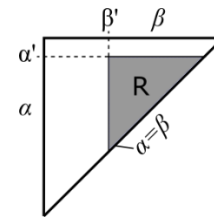


Figure 5: Region R over which the  $P(\alpha, \beta)$  is integrated to compute the Everett function  $E_V(\alpha', \beta')$

The use of this Everett function allows the analytical computation of the Preisach model without numerical integration or a table lookup, thus considerably improving the performance of the developed Preisach flux tube model. The implemented analytical form of the Everett function can be adjusted with seven parameters and thus covers only a limited but a wide range of possible hysteresis shapes.

### 2.3 Eddy Currents

In addition to the previously described static hysteresis models this section explains modeling of dynamic hysteresis, i.e. consideration of eddy currents in a ferromagnetic core. An approach proposed in [5] has been implemented. The total magnetic field strength  $H(t)$  of an hysteresis element consists of a static  $H_{\text{stat}}(t)$  and a dynamic component  $H_{\text{eddy}}(t)$ :

$$H(t) = H_{\text{stat}}(t) + H_{\text{eddy}}(t) \quad (6)$$

$H_{\text{stat}}(t)$  results from the static Preisach or Tellinen hysteresis model.  $H_{\text{eddy}}(t)$  is computed as product of the classical eddy current factor  $\sigma_{\text{cl}}$  [5, 7] and the change of the magnetic flux density in the core.

$$H_{\text{eddy}}(t) = \sigma_{\text{cl}} \cdot \frac{dB}{dt} \quad (7)$$

$\sigma_{\text{cl}}$  results from a homogenization approach. It considers the real electrical conductivity  $\sigma$  and the thickness of individual steel sheets for computation of an adapted conductivity for a whole stack of laminations:

$$\sigma_{\text{cl}} = \frac{\sigma \cdot d^2}{12}. \quad (8)$$

Figure 6 shows exemplarily a simulated hysteresis loop together with its ferromagnetic and its eddy current components. The simulated network model is similar to the one shown in Figure 14.

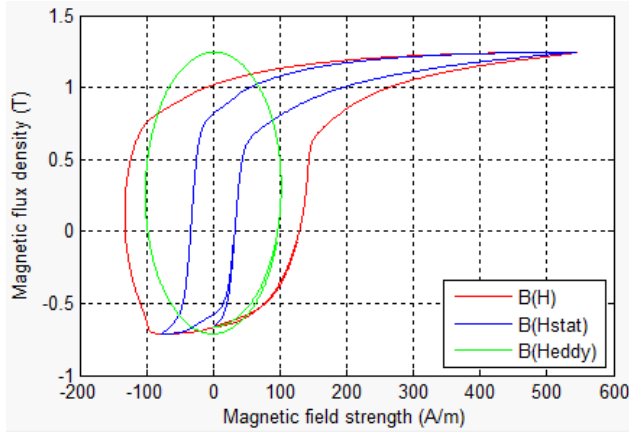


Figure 6: Simulated biased hysteresis loop  $B(H)$  with its ferromagnetic component  $B(H_{\text{stat}})$  and eddy current component  $B(H_{\text{eddy}})$  for a sinusoidal voltage excitation at 400 Hz

## 2.4 Thermal Losses

Continuous computation of the hysteresis and its static and dynamic components allows for accurate computation of the instantaneous hysteresis losses. The total power loss  $P$  consists of the static hysteresis loss  $P_{\text{stat}}$  and the eddy current loss  $P_{\text{eddy}}$  and can be computed as follows:

$$P = P_{\text{stat}} + P_{\text{eddy}} = H(t) \cdot \frac{dB(t)}{dt} \cdot V. \quad (9)$$

$V$  denotes the core volume. For  $P_{\text{stat}}$  and  $P_{\text{eddy}}$  the equations (10) and (11) apply:

$$P_{\text{stat}} = H_{\text{stat}}(t) \cdot \frac{dB(t)}{dt} \cdot V, \quad (10)$$

$$P_{\text{eddy}} = H_{\text{eddy}}(t) \cdot \frac{dB(t)}{dt} \cdot V = \sigma_{\text{cl}} \cdot \left( \frac{dB(t)}{dt} \right)^2 \cdot V. \quad (11)$$

According to the hysteresis plot of Figure 6, Figure 7 shows the simulated time courses of the magnetic field strength and the core losses together with their static and dynamic components. The function of the magnetic core as an energy storage is easy to recognize in Figure 7 (b). When the loss power is negative, energy is fed back into the system. Furthermore, the instantaneous computation of the power losses is advantageously, because the losses are computed correctly independent of the applied frequencies and waveforms. The total hysteresis losses can be passed to a conditional heat port as a heat flow. This allows for simple integration of the developed hysteresis elements into a thermal network model.

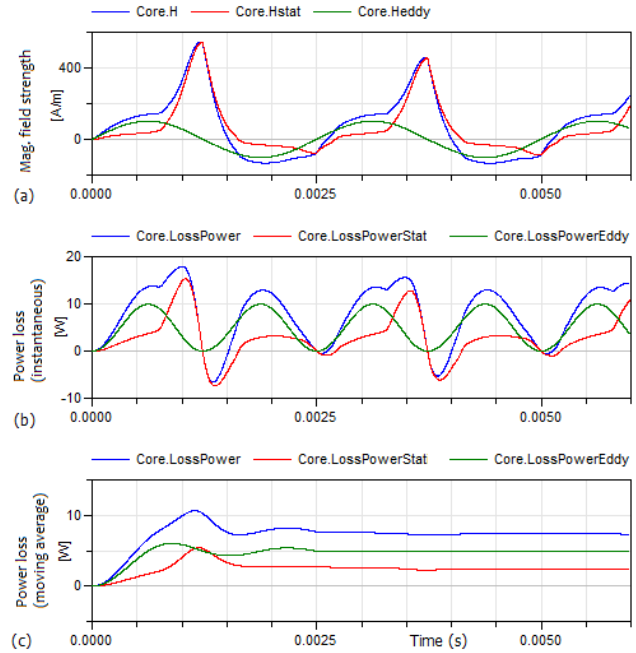


Figure 7: Simulated time course of the magnetic field strength (a), the instantaneous hysteresis losses (b) and the averaged hysteresis losses (c) of a magnetic core element excited with a sinusoidal voltage at 400 Hz (according to the hysteresis plot of Figure 6).

## 3 New Components in the Library Modelica.Magnetic.FluxTubes

The FluxTubes library extension mainly consists of the developed hysteresis elements for modeling of ferromagnetic materials including permanent magnets, an extended materials package and a new components package with models of one- and three-phase transformers.

### 3.1 Package HysteresisAndMagnets

The new hysteresis elements are grouped together in the package FluxTubes.Shapes.HysteresisAndMagnets (see Figure 8). There are four different Tellinen hysteresis models. The difference between these models is the definition of the static hysteresis behavior. The elements CuboidHystTellinenSoft and CuboidHystTellinenHard use very simple hyperbolic tangent shape functions to describe the envelope curve of the hysteresis directly. The shape functions are tailored to the description of soft and hard magnetic materials and can be easily adjusted with four meaningful parameters only (e.g. for the soft magnetic shape functions: saturation magnetization, coercivity, remanence and a multiplier for the vacuum permeability to adjust the slope in the saturation re-



gion). The elements `CuboidHystTellinenEverett` and `CuboidHystPreisachEverett` use the analytical description of the Everett function presented in section 2.2. Several parameter sets to adjust the material behavior are available in the material package `HysteresisEverettParameter`. An additional option to specify the hysteresis shape offers the element `CuboidHysteresisTellinenTable`. This element uses table data for the rising and falling branches of the limiting hysteresis loops, and thus allows for the definition of almost arbitrary hysteresis shapes. Two additional elements for modeling permanent magnets are included in the package: the `CuboidLinearPermanentMagnet` for linear hard magnetic materials defined in `Material.HardMagnetic` and the `CuboidHystTellinenPermanentMagnet` which also accounts for hysteresis and thus also for demagnetisation processes of a permanent magnet.

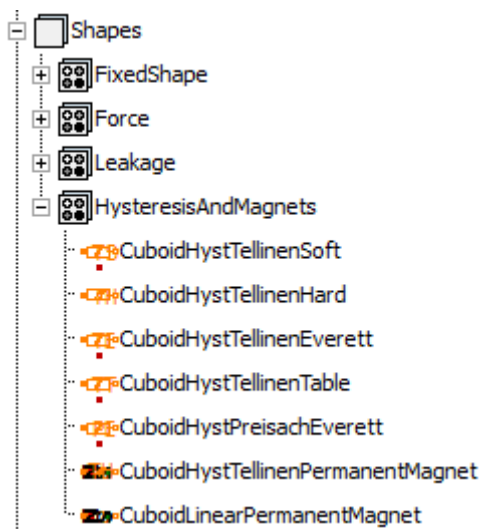


Figure 8: New flux tube elements of package `Shapes.HysteresisAndMagnets`

Most of these elements also allow for consideration of eddy currents and provide a conditional heat port, which simplifies delivery of the generated hysteresis losses to a thermal network model.

### 3.2 Package Components

Based on the hysteresis elements described above, package `Components` contains predefined models of single- and three-phase transformers of different topologies (See Figure 9). The transformer models can be widely configured (e.g. core geometry, winding parameters, stray flux, material) and thus easily be adapted to specific needs. Consideration of both static hysteresis and eddy currents allows for accurate computation of losses, saturation effects, inrush currents and frequency behavior. Again, a conditional

heat port allows for connection of the magnetic model to a thermal network model. Besides the hysteresis losses also the copper losses of the windings are automatically considered.

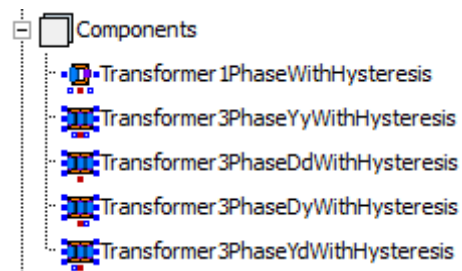


Figure 9: The new `Components` package of `Modelica.Magnetic.FluxTubes`.

### 3.3 Extension of Package Material

For a simple adaption of the developed hysteresis models to real magnetic materials two new material packages have been included in the library extension: the package `HysteresisEverettParameter` and the package `HysteresisTableData`. Since accurate hysteresis data is hardly available, a measurement setup has been developed and built for hysteresis characterization of magnetic materials according to the DIN EN 60404-2/4/6. Figure 10 shows the block diagram of the utilized measurement setup. The 25 cm Epstein frame used for the characterization of 30 mm x 280 mm magnetic steel sheet samples is shown in Figure 11.

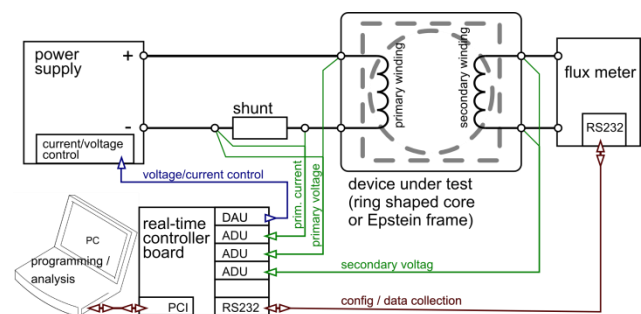


Figure 10: Block diagram of the hysteresis measurement setup

The setup allows hysteresis measurements within a range of the magnetic field strength of  $\pm 10$  kA/m. For the implementation of package material the static hysteresis characteristic of several steel sheet samples has been determined. Three exemplary hysteresis loops for three different steel sheet qualities are shown in Figure 12. For verification of the developed hysteresis models also dynamic loops have been measured.

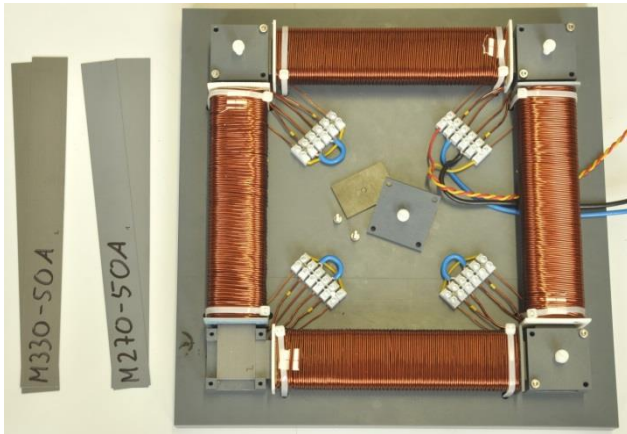


Figure 11: Built-up 25 cm Epstein frame used for hysteresis characterization of magnetic steel sheets

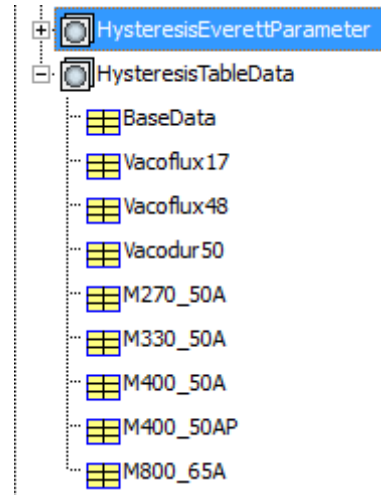


Figure 13: Current content of the hysteresis materials library of Modelica.Magneti.FluxTubes

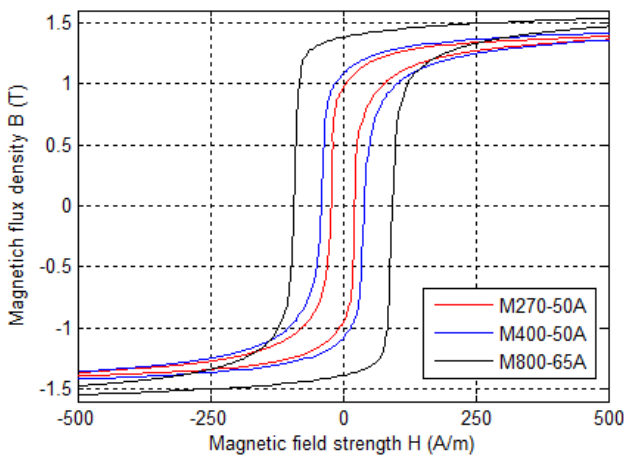


Figure 12: Exemplarily measured hysteresis loops of three different steel sheet qualities

In package HysteresisTableData the measured hysteresis envelope curves are directly stored as table data. The data provided in this package can be used with the CuboidHystTellinenTable hysteresis element. Since the hysteresis elements CuboidHystTellinenEverett and CuboidHystPreisachEverett both need Everett function parameter sets for their configuration (see section 2.2), the package HysteresisEverettParameter contains such predefined parameter sets. They have been identified using the hysteresis loops of package HysteresisTableData as reference. Figure 13 shows the content of package HysteresisTableData. In Addition to the measured hysteresis data some data of cobalt-iron-alloys [10] have also been included in the library. It is planned to extend the material package when new data are available.

## 4 Verification Results

To verify the correct simulation of the developed hysteresis models, several experiments have been carried out to compare the model behavior to measurements and to steel sheet datasheets.

### 4.1 Static and Dynamic Hysteresis Loops

In a first experiment the Epstein frame of Figure 11 was used to measure total hysteresis loops (superposition of static and dynamic loops) at several excitation frequencies.

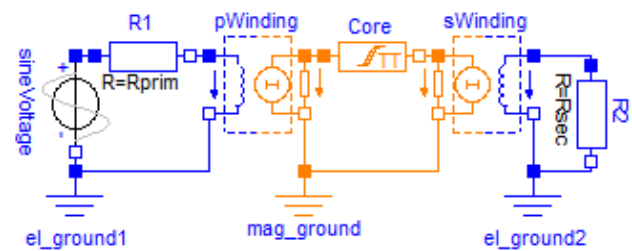


Figure 14: Simple model of the measurement setup including the Epstein frame shown in Figure 11

In addition, the measurement setup has been modeled with a simple model consisting of a sine voltage source, a series resistance, a primary and secondary winding and the magnetic core, latter being modeled with element CuboidHystTellinenTable (see Figure 14). The stray flux of the magnetization coils is considered within the element pWinding.

Figure 15 shows the measured and the simulated  $B(H)$  loops of the magnetic core for identical excitations at several frequencies. In principal, a good agreement can be seen. The deviations in the 200 Hz loops are likely due to neglect of excess eddy currents, which are not considered in the hysteresis models.

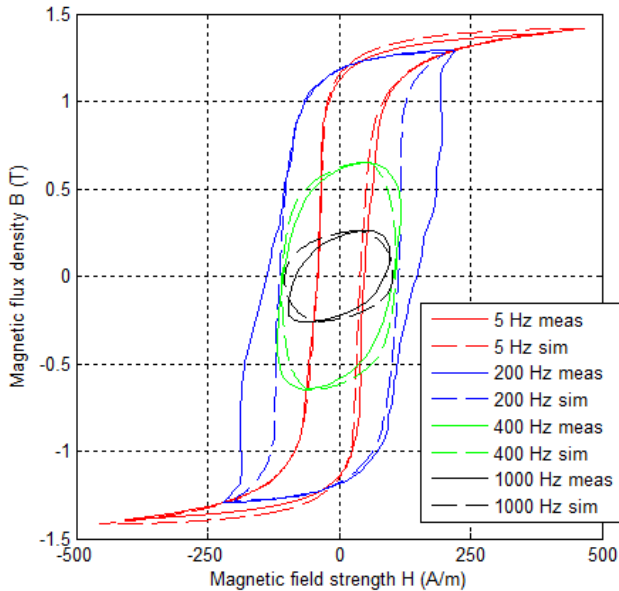


Figure 15: Comparison of measured and simulated total  $B(H)$  loops using M330-50A steel sheets and the Cu-boidHystTellinen hysteresis element for modeling of the core

## 4.2 Time Course of Current and Power Losses

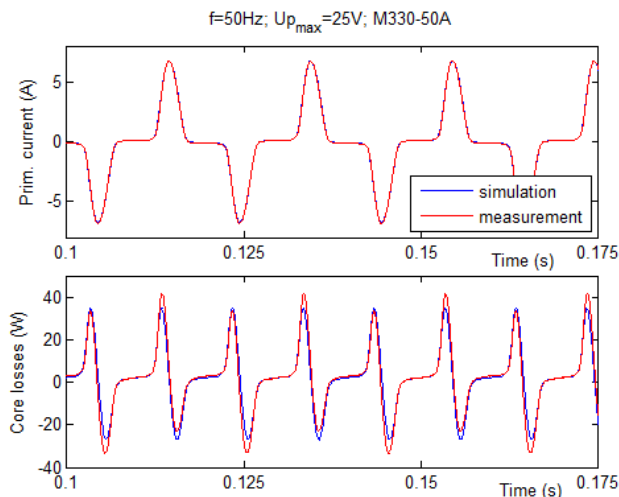


Figure 16: Comparison of measured and simulated primary current and core losses of a transformer excited by a sinusoidal primary voltage of  $U_{pmax}=25$  V and  $f=50$  Hz.

In a second experiment a similar setup and model was used to compare the time course of the primary current and the iron losses. This has been done for a highly saturated core at an excitation frequency of  $f=50$  Hz and a primary voltage magnitude of  $U_{pmax}=25$  V (see Figure 16) and for a moderately saturated core at  $f=100$  Hz and  $U_{pmax}=40$  V (Figure 17).

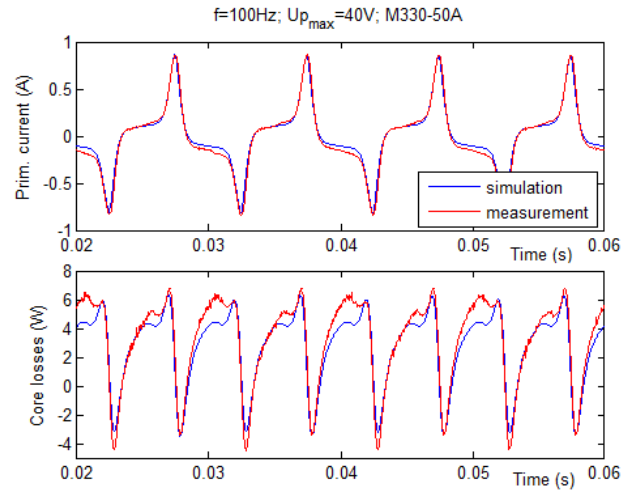


Figure 17: Comparison of measured and simulated primary current and core losses of a transformer excited by a sinusoidal primary voltage  $U_{pmax}=40$  V and a frequency  $f=100$  Hz

## 4.3 Specific Core Losses

In a third experiment the model of Figure 14 was used again. The sinusoidal primary voltage has been gradually increased at a fixed frequency of 50 Hz for two different core materials. For each step the peak value of the magnetic flux density as well as the average power loss of the core have been recorded. Based on the average power loss, the core volume and the density of the core material the specific total losses of the core material have been evaluated. Figure 18 shows the comparison of these values to manufacturer data, which was extracted from the manufacturer datasheets of the investigated materials [11, 12]. Again, a very good agreement over a wide range of magnetic excitations up to saturation is evident.

## 5 Summary

Within the Clean Sky project MoMoLib an extension of the library Modelica.Magnetic.FluxTubes has successfully been implemented. The extension mainly consists of hysteresis elements for modeling of ferromagnetic and dynamic hysteresis of magnetic

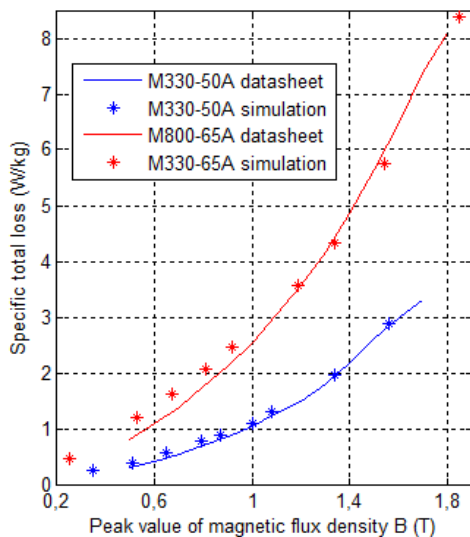


Figure 18: Comparison of specific core losses between data extracted from manufacturer datasheets [11, 12] and simulation results for two different steel sheet qualities M330-50A and M800-65A

materials during transient simulation of electromagnetic components with lumped network models. Two different hysteresis models, the Tellinen and the Preisach model have been implemented. Their static hysteresis characteristics can be parameterized in three different ways: with simple tanh() shape functions, hysteresis table data or with parameter sets of an analytical Everett function. A material package, mainly based on in-house measurements, provides the hysteresis data of several magnetic materials for easy parameterization of the models. Dynamic hysteresis is considered with a  $dB/dt$  term. The consideration of static and dynamic hysteresis during transient simulation allows for accurate determination of hysteresis losses. This becomes more and more important during design of electromagnetic actuators and systems due to increasing requirements in terms of power density and miniaturization. A series of experiments showed the correct behavior and the accuracy of the developed hysteresis elements. Additionally, based on the hysteresis elements, models for permanent magnets as well as single- and three-phase transformer models have been implemented.

## 6 Acknowledgement

The authors would like to thank the Clean Sky Joint Technology Initiative for funding of the presented work within Project No. 296369 MoMoLib “Modelica Model Library Development for Media, Magnetic Systems and Wavelets”. Additionally, the authors wish to thank Prof. Martin Otter of the German

Aerospace Center (DLR) for his valuable support regarding numerical efficiency of the developed hysteresis models.

## References

- [1] Modelica Association, Modelica Standard Library, <https://www.modelica.org/libraries/-Modelica> (May 11, 2012).
- [2] T. Bödrich and T. Roschke, A Magnetic Library for Modelica, in Proc. of the 4th International Modelica Conference, 2005, pp. 559–565.
- [3] T. Bödrich, Electromagnetic Actuator Modelling with the Extended Modelica Magnetic Library, Proc. of 6th Int. Modelica Conf., Bielefeld, Germany, March 3-4, pp. 221–227, 2008.
- [4] J. Ziske, Magnetic Hysteresis Models for Modelica, Proc. of 9th Int. Modelica Conf., Munich, Germany, September 3-5, pp. 151-158, 2012
- [5] J. Tellinen, A Simple Scalar Model for Magnetic Hysteresis, IEEE Transactions on Magnetics, vol. 24, no. 4, pp. 2200 – 2206, July 1998.
- [6] F. Preisach, Über die magnetische Nachwirkung, Zeitschrift für Physik A Hadrons and Nuclei, vol. 94, pp. 277–302, 1935.
- [7] A. J. Bergqvist and S. G. Engdahl, A Homogenization Procedure of Field Quantities in Laminated Electric Steel, Magnetics, IEEE Transactions on , vol.37, no.5, pp.3329-3331, Sep 200
- [8] I. Mayergoyz, Mathematical Models of Hysteresis and their Application. Elsevier, 2003.
- [9] T. Yamaguchi and F. Ueda and E. Yamamoto, Simulation of Hysteresis Characteristics of Core Materials Using the Everett Function, Magnetics, IEEE Transactions on , vol. 4, no.6, pp.353-359, Jun 1989
- [10] Soft Magnetic Cobalt-Iron-Alloys, Vacuumschmelze GmbH, 2001, [http://www.vacuumschmelze.com/fileadmin/docroot/medialib/-documents/broschue-ren/htbrosch/Pht-004\\_e.pdf](http://www.vacuumschmelze.com/fileadmin/docroot/medialib/-documents/broschue-ren/htbrosch/Pht-004_e.pdf) (May 5th, 2013).
- [11] ThyssenKrupp Steel AG: Power Core M800-65A, 2009. [http://www.thyssenkrupp-steel-europe.com/-upload/binarydata\\_tksteel05d4cms/59/90/78/02/00/00/2789059/Werkstoffblatt\\_M800-65A.pdf](http://www.thyssenkrupp-steel-europe.com/-upload/binarydata_tksteel05d4cms/59/90/78/02/00/00/2789059/Werkstoffblatt_M800-65A.pdf) (Dec. 6<sup>th</sup>, 2013)
- [12] ThyssenKrupp Steel AG: Power Core M330-50A, 2009. [http://www.thyssenkrupp-steel-europe.com/-upload/binarydata\\_tksteel05d4cms/45/90/78/02/00/00/2789045/Werkstoffblatt\\_M330-50A.pdf](http://www.thyssenkrupp-steel-europe.com/-upload/binarydata_tksteel05d4cms/45/90/78/02/00/00/2789045/Werkstoffblatt_M330-50A.pdf) (Dec. 6<sup>th</sup>, 2013)

# Custom Annotations: Handling Meta-Information in Modelica

Dirk Zimmer<sup>1</sup>, Martin Otter<sup>1</sup>, Hilding Elmqvist<sup>2</sup>, Gerd Kurzbach<sup>3</sup>

<sup>1</sup>German Aerospace Center (DLR), Institute of System Dynamics and Control,  
D-82234 Wessling, Germany,

<sup>2</sup>Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

<sup>3</sup>ITI GmbH, 01067 Dresden, Germany

[dirk.zimmer@dlr.de](mailto:dirk.zimmer@dlr.de), [martin.otter@dlr.de](mailto:martin.otter@dlr.de), [hilding.elmqvist@3ds.com](mailto:hilding.elmqvist@3ds.com), [kurzbach@ititim.com](mailto:kurzbach@ititim.com)

## Abstract

Annotations and attributes form an important part of the Modelica language. They are used to include various meta-information such as documentation, external C-code, compilation hints, etc. Given the increasingly wide field of potential applications the set of useful annotations becomes too large to be included in the language specification. Hence we present a proposal how a Modelica modeler may define his own annotations and how such custom annotations can be organized within Modelica libraries. In the long term, the goal is to move the definition of standardized annotation, as well as of attributes, from the Modelica specification to a standard library.

*Keywords: meta-information; custom annotations; optimization setup; Monte Carlo simulation setup; Kalman filter setup; uncertainty setup.*

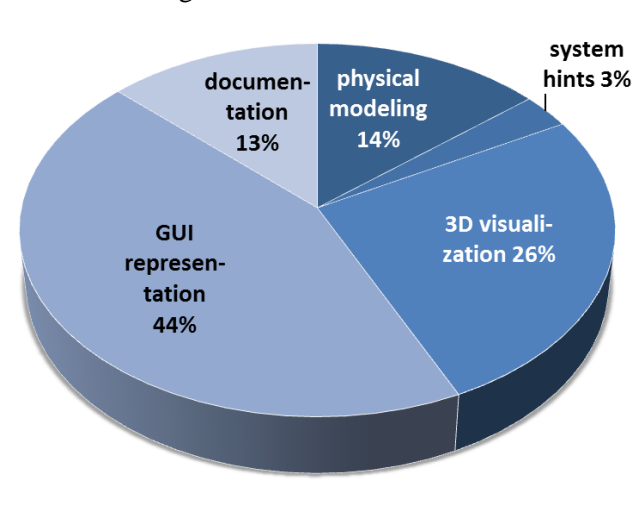
## 1 Introduction

The main purpose of Modelica is to enable the equation-based modeling of physical systems. In addition to this primary objective, the modeler has to care about the usability of his/her components. This includes a variety of tasks: documentation needs to be written, icons need to be drawn, a 3D visualization has to be provided, and compilers might need hints for generating more efficient code.

All this is meta-information to the actual physical model but as Figure 1 shows, it can account for a major share of the code: For the FixedTranslation component (a rigid rod in 3D Mechanics), the physical modelling contributes only to 14% of all the code. Of course such a comparison is skewed since it is doubtful to compare manually typed equations with auto-generated code for graphical objects but nevertheless the handling of meta-information deserves to be a major concern for the future design of the Modelica language.

An improved solution for meta-information in Modelica becomes necessary since there is a desire to include more and more information into the models. Especially, a model might be used not only in simulation, but in other analysis and synthesis methods, and then additional model-specific data is needed. For example, sensitivity analysis needs uncertainty data for model variables, Monte Carlo simulation needs stochastic distribution data on states and/or parameters, an optimization setup needs the information which parameters and/or input signals shall be optimized, and in which range the optimization shall take place.

To meet these demands, we propose an enhancement to Modelica: custom annotations. But before we address the new proposal, let us look at the current handling of meta-information in Modelica and its weak spots and then formulate the requirements for a new design.



**Figure 1:** Percentage of characters devoted to certain tasks in Modelica.Mechanics.MultiBody.Parts.FixedTranslation. Source: (Zimmer 2008)

## 2 Handling of Meta-Information

### 2.1 Meta-Information in Current Modelica

The Modelica language (*Modelica Association 2012*) offers currently two devices that are used for meta-information: annotations and attributes.

Standard annotations are defined for a multitude of issues: graphical information for icons and diagrams, GUI-design, documentation, version handling, etc. Here is a typical annotation in a Modelica code. It describes the representation of a parameter in the GUI of the modeling environment and advises the compiler to evaluate this parameter before generating code:

```
parameter RotationSequence sequence
  "Sequence of rotations "
  annotation(Evaluate=true,
             Dialog(tab="Advanced",
                   enable=not useQuaternions)
             );
```

Since the information is mostly of no interest for the human reader and an inadvertent manipulation shall be prevented, most modeling environments for Modelica hide annotations from the user by default.

Attributes are also used for meta-information although this information is mostly linked closer to the physical variables (or parameters) of the model: physical units, minimum and maximum boundaries or potential start values for iterative solvers are described by this language construct. The following example contains attributes for the start value, whether they are used for as initial equations, and whether the variable shall be used as state-variable depending on another parameter.

```
SI.AngularVelocity w_a[3] (
  start=Frames.resolve2(R_start,w_0_start),
  fixed=fill(w_0_fixed, 3),
  each stateSelect=
    if enforceStates then
      (if useQuaternions then
        StateSelect.always
      else StateSelect.never)
    else StateSelect.avoid)
```

The two listings above give a quick glance on how meta-information is stored within Modelica. The current solution served fine for more than a decade but it has come to its limitations. We are confronted with two major weaknesses: rising complexity and ambiguity.

The first weakness is simply the sheer amount of definitions that are needed. The current version of the specification devotes already 20 pages for more as 70 annotations and roughly 7 pages for about 10 attributes. The specification is already a long document and further inflation must be prevented. Also

we have to keep in mind that the specification is primarily targeted for tool vendors and not for end-users. Most end-users should not have to consult the specification but rather refer to other material.

The second weak point is that the definition in the specification is often not complete. For example, it is usually not defined on which elements an annotation can be placed and only from context one might deduce that annotation “Evaluate” makes sense only for primitive data types, whereas annotation “documentation” might make sense at many places, but is actually in use only on classes and not on components.

The third weak spot is the ambiguity between the two different concepts. Whether some information belongs to an attribute or to an annotation is not always clear and has often be a discussion point in the design process of the language. For example, `stateSelect` is an attribute used to tell the compiler which variables shall form the state-space of the model. `Evaluation` is an annotation and used to tell the compiler which parameters to evaluate beforehand.

Such discussions are often influenced by the differences in which way attributes and annotations can be accessed. Attributes can be set in (even nested) modifiers, annotations cannot. Vendors can specify their own annotations but they are not allowed to do this for attributes.

### 2.2 Meta-Information in other Languages

In (*Zimmer 2008*) the handling of meta-information (here denoted as multi-aspect modelling) is discussed for various other modeling languages such as VHDL-AMS or SPICE3. Then another approach is proposed based on the experimental language Sol (*Zimmer 2009*). Here the modeler is given the opportunity to define his/her own annotations by means of environment packages and then can use them by instantiating the components of this package within pre-specified sections of his model. This is conveniently possible because in Sol components have first-class status (*Burstall and Strachey, 2000*) unlike in current Modelica.

Another (although similar) proposal is discussed in (*Zimmer, 2012*). It is based on another experimental language called Hornblower. Also here annotations can be defined within packages and then used within the models. This concept treats annotations like “loosely attached parameters”. These are parameters that can be set but do not have to be set. This is possible since such parameters were ensured to always have a reference to a default object.

Meta information can also be defined in certain programming languages, especially in Java (Coward 2004): From [http://en.wikipedia.org/wiki/Java\\_annotation](http://en.wikipedia.org/wiki/Java_annotation): “Classes, methods, variables, parameters and packages may be annotated. Unlike Javadoc tags, Java annotations can be reflective in that they can be embedded in class files generated by the compiler and may be retained by the Java VM to be made retrievable at run-time. It is possible to create meta-annotations out of the existing ones in Java”.

General programming languages often cope much better with meta-information than declarative modeling languages because they own suitable data-structures and often contain already sufficient means for introspection. In Python, there are doc-strings for documentation but they are just predefined class members. Also class or function decorators are used to express a meta-construct on an item, but also these constructs are regular language constructs. In Python there is no need to have language constructs solely devoted to meta-information; instead the regular constructs prove to be sufficient. This shows to us that it is a good idea to reuse regular language constructs for meta-information in Modelica as much as reasonable feasible.

### 3 Design Goals

In concrete terms, the following goals shall be reached:

- The modeler must be able to define annotations by him- or herself.
- Existing annotations or attributes shall be defined in the same way and removed from the specification (at least as many as possible). This will require to introduce more powerful data structures in Modelica.
- The annotations shall be organized in packages so that an end-user can browse through them and do not need to address the specification anymore.
- The modeler must be able to apply custom annotations in (nested) modifiers.
- Annotation must never be required to be provided by the user. Annotations and its parts are always meta-information that can be given optionally.
- The proposed design must be backwards compatible so that current code is not broken.
- The proposed design can make some language constructs obsolete that can then be removed from the language in the future.

- It must be specified how the meta-information contained in custom annotations is handled for model-export (for instance FMI).

Based on these goals, we have developed a suitable design for a future version of the Modelica language: *Custom annotations*.

## 4 Design Proposal

The basic idea of our proposal is to use basic Modelica “records” to define custom annotations and then make them better applicable by enabling the use of annotations in modifiers. In this way, new features can be introduced without having to define many new language elements. Note, records are also the basis of nearly all built-in Modelica annotations. A similar concept in (Zimmer, 2012) served as additional starting point. However the transition from an experimental language to a heavily applied language like Modelica demanded several adaptations.

### 4.1 Use of Records within Annotations

Let us look at an example: Here the Modelica package `OptimSetup` shall be used to define an optimization setup for a model and contains the definition of three record classes that each can be applied as custom annotations.

```
package OptimSetup
  record Tuner "Parameter to be optimized"
    parameter Boolean active = true
    "= true, if parameter is optimized";
    parameter Real min
    "Optional minimum value";
    parameter Real max
    "Optional maximum value";
  end Tuner;

  record Minimize
    "Signal that should be minimized"
    parameter Boolean active = true
    "= true, if used as criterion";
    parameter Real demand = 1.0
    "Value/demand is minimized";
  end Minimize;

  record OptimOptions "Default options for
    the optimization setup"
    parameter String method
    "Optional optimizer method";
    parameter Real tolerance = 0.001
    "Tolerance of optimization";
  end OptimizationOptions;

  record SimOptions
    "Default options for simulation setup"
    parameter String method;
    parameter Real stopTime;
  end SimulationOptions;
end OptimSetup;
```

The record and type definitions of package `OptimSetup` can now be used to describe the setup of an optimization. For example, a parameter can be marked to be a “Tuner” that shall be optimized. The following statement

```
parameter Real p1
  annotation(OptimSetup.Tuner(
    active=true, min=-2));
```

indicates that for a default optimization setup for this model, parameter `p1` shall be used as tuner and shall have a constraint `p1 >= -2`. A custom annotation is identified by the full path name of the `Tuner` record class. This defines a new instance of the record, together with a modifier on this record. So, conceptually, this custom annotation is equivalent to the following declaration:

```
OptimSetup.Tuner name(active=true,
  min=-2);
```

and the name of the instance is not defined, because not needed (the identification of the data is via the class name). Exactly in the same way as in a standard declaration, an element of a record needs not to have a default value or a binding equation in its class, and also not in a modifier.

The lookup of a class name inside an annotation is performed on global scope, so always full path names must be given. This simplifies and speeds up the lookup, especially once built-in annotations are defined as custom annotations in a second phase<sup>1</sup>.

A custom annotation can be also defined on a class. Furthermore, custom and built-in annotations can be within the same annotation declaration, by using a comma-separated list as usual. For example:

```
model ControlledDrive
  ...
  annotation (Documentation(info="..."),
    OptimSetup(
      OptimOptions(tolerance=1e-3),
      SimOptions(stopTime=4.0, tol=1e-6)
    ));
end ControlledDrive;
```

## 4.2 Enabling Annotations within Hierarchical Modifiers

So far the only extension to the current Modelica language has been that regular Modelica records can be used within annotations. Taken for itself, this is already a progress but it does not suffice to provide the desired level of functionality. For many applica-

tions it is important that annotations can be applied within hierarchical modifiers.

Hence we propose to enable the use of annotation statements within hierarchical modifiers. Here, custom annotations can be either newly constructed or an already defined custom annotation can be modified. Let us look at two corresponding examples:

```
MyCar car(p1 annotation(
  OptimSetup.Tuner(active=false)),
  p2 annotation(
    OptimSetup.Tuner(max=4)),
  p3(min=-3) = 5 annotation(
    OptimSetup.Tuner(min=-2, max=3)
  ));
```

In this example, the already defined `Tuner.active` value of `p1` is modified to `false`. Parameters `p2` and `p3` are assumed to be defined in `MyCar` without any annotation. The declarations above introduce new instances of custom annotation `OptimSetup.Tuner`, and modify these instances with the given values.

Whereas in principle built-in annotations could be applied within hierarchical modifiers too, we propose to restrict this in a first phase because built-in annotations operate on data structures that are unavailable as the standard language elements and then a standard modifier cannot be applied. This is for instance the case for the annotations describing icons that use case records as data elements. For the future, one may solve this problem by enriching Modelica with suitable data structures. This is a topic where discussion is ongoing in parallel.

As with built-in annotations, it is not possible to read and/or use the value of a custom annotation in a Modelica class (only the translator can use the information contained within annotations by either performing appropriate actions or by passing them to its backend).

## 4.3 About the Use of Hierarchical Records

Since regular Modelica records can be used within annotations, this holds also true for hierarchical records. This is per se not problematic but a few details require a discussion.

Let us suppose we add a hierarchical record `OptimSetup` to our previously present `OptimSetup` package:

```
package OptimSetup

  record Tuner [...]

  record Minimize [...]

  record OptimOptions [...]
```

<sup>1</sup> When built-in annotations are defined with custom annotations, it is proposed that they are placed, e.g., in `ModelicaServices.Annotations`, and that this package is inspected first and then the global scope.



```

record SimOptions [...]

record Options
  OptimOptions optOpts =
    OptimOptions(method="sqp");
  SimOptions simOpts;
end Options;

end OptimSetup;

```

The following example model shows a correct way of using the hierarchical record and a wrong way to do it:

```

model ControlledDrive
  ...
  // correct Modelica code
  annotation(OptimSetup.Options(
    optOpts = OptimSetup.OptimOptions(
      method="pattern"),
    simOpts(stopTime = 2)
  ));

  // wrong Modelica code
  annotation(OptimSetup.Options(
    optOpts(method="pattern"),
    simOpts = OptimSetup.SimOptions(
      stopTime=2)
  ));
end ControlledDrive;

```

The record `Options` has two element record instances:

The first element `optOpts` is defined with a record constructor. This means there is a binding equation to `optOpts`. Binding equations cannot be modified with a modifier. They can only be replaced by another binding equation. Therefore in the “correct” code, a record constructor is used to define modified elements. Note, a record constructor must return a complete record, and therefore all elements of the record must have a value (either defined in the constructor or the default values from the class).

The second element `simOpts` is defined without a default value or a binding equation. When instantiating it in the “correct” code, a modifier to its elements is given. In this case, not all elements must be modified. In the “wrong” code, a record constructor is used to define modified elements. This would be fine, but element `method` has no default value in the `SimOptions` class, and the record constructor has no input argument for this element, and this is then an error.

Please note that all this is already standard Modelica semantics and holds for standard record declarations, and therefore it shall hold for record declarations in a custom annotation as well. We have just repeated these points for the sake of clarity.

#### 4.4 On Inherited Elements

Since models can be inherited, both the superclass and the subclass may define the same custom annotation. Two cases need to be distinguished:

**Case 1:** the additional custom annotation is defined in the **extends clause**, such as:

```

model MyCar
  extends Car(
    p1 annotation(OptimSetup.Tuner(
      active=false)),
    annotation(OptimSetup.OptimOptions(
      tolerance=1e-3))
  );

```

It seems natural to handle this case as modifier, if the custom annotation was already defined in one of the superclasses (otherwise, a new custom annotation is introduced). Therefore, the model can contain the same custom annotation on one element at most once, and the semantics is well-defined (the semantics of a Modelica modifier).

**Case 2:** the additional custom annotation is defined as **class annotation**, such as:

```

model MyCar
  extends Car;
  annotation(OptimSetup.OptimOptions(
    tolerance=1e-4));
end MyCar;

```

In this example it is assumed that in model `Car` the element `tolerance` is defined as `1e-3` and in model `MyCar` as `1e-4`. This creates an ambiguity that needs to be resolved. The Modelica built-in annotations use two different semantics: For the “documentation” annotation, all superclass definitions are ignored. For the “Icon” and “Diagram” annotations, the subclass and superclass definitions are applied after each other (so all of them have an effect).

It is hence proposed to support all these forms by allowing that the same class custom annotation can be used in super- and subclasses. The consequence of this is that an annotation may occur several times in a class. A vector of records can be used to represent the multiple occurrences of one annotation in one class. The order of the vector elements thereby corresponds to the order of inheritance. In case of multiple inheritance, the order of the corresponding `extends` statements is used.

When generating the standardized “output” format of custom annotations in case 2, such definitions can be represented as a vector of records. The target tool that is using this custom annotations has then to decide how to comprise the information contained in all vector elements. In general, useful strategies are:

- Only utilize the latest definition (last element of the vector), and ignoring definitions in superclasses.
- Only utilize the latest definition (last element of the vector), and triggering an error, if equivalent definitions are provided in superclasses (prior elements of the vector).
- Utilizing all definitions by merging them together (following the order of the vector).

## 4.5 Remaining Issues

### 4.5.1 Handling of undefined annotations using parameters

One problem arises from the point that it is often not possible to define a meaningful default value for a custom annotation element. For example, “method” of the optimization options defines the optimization method to be used and this depends on the optimization environment where the model and the custom annotations will be imported. It is therefore not possible to define a meaningful default value. Furthermore, the modeler may decide to not define a value for the method in the custom annotation of the Modelica model. For this reason, not all elements of a custom annotation record needs to have a value when it is actually used in an annotation. If an element, such as “method” is not explicitly set, then it is not present in the annotation and this implicitly means that the target environment has to cope with this undefined value in a target specific way.

This is not an issue as long as custom annotations are specified directly in the textual layer but providing custom annotations in this way is often not convenient for a user. Instead special blocks or models might be defined where all the relevant information of the custom annotation can be defined in a menu. For example, the `OptimOptions` above might be defined as a partial model that is used via inheritance:

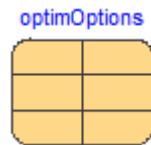
```

partial model OptimOptions
  OptimSetup.OptimOptions optimOptions
  annotation(Placement(...));
  annotation(OptimSetup.OptimOptions(
    method =optimOptions.method,
    tolerance =optimOptions.tolerance),
    Diagram(...));
end OptimOptions;

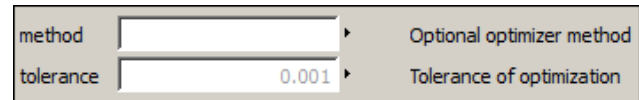
model DrumBoiler // shall be optimized
  extends OptimOptions;
  ...
end DrumBoiler;

```

In the diagram layer of `DrumBoiler`, the `optimOptions` are present with a record icon:



and clicking on this icon opens a convenient parameter menu for the definition of the options:



This approach is welcomed by the user, but currently involves one severe drawback: the user has to provide a value also for optional elements, such as `method`, because this element is propagated to the annotation (above: `OptimSetup.OptimOptions (method = optimOptions.method)`). If no value would be provided, then the translator needs to trigger an error. Therefore, the user can no longer express to not define such an optional custom annotation.

For this reason, it is considered to introduce a limited form of “undefined” handling of definitions and modifiers that is handled during the translation process. The goal is

- to remove annotations if they are defined with “undefined” elements.
- to remove modifications performed in a lower hierarchical level.

So far several approaches to this problem have been suggested but a sufficient level of maturity has yet to be reached.

### 4.5.2 Restricting the application of annotations to certain types

In the current proposal, any annotation might be applied to anything. Also meaningless applications are allowed. For instance the annotation for an optimization tuner cannot only be applied to real numbered parameters but also to Boolean parameters or strings. Even an application to a model class is allowed. Also the annotation for the simulation setup cannot only be applied to model classes as originally intended but also to individual components or variables where it becomes meaningless.

Taken for themselves, such meaningless or ill-applied annotations are not harmful since the metadata cannot corrupt the main code but yet it might be better to restrict the applicability to annotation to give a better guidance where an annotation is supposed to be used.

How such a restriction is best imposed or if it shall be imposed at all, is still open for discussion. One way of doing it, could be to express the restriction by

annotations themselves in the corresponding annotation record. A new built-in annotation `AnnotationTarget` could serve this purpose. It would contain an enumeration listing for the various possibilities where an annotation can be applied:

```
record AnnotationTarget
  type Target = Enumeration(
    Any,
    ClassDefinition,
    ModelDefinition,
    BlockDefinition,
    ConnectorDefinition,
    RecordDefinition,
    FunctionDefinition,
    TypeDefinition,
    ComponentDeclaration,
    ModelDeclaration,
    BlockDeclaration,
    ConnectorDeclaration,
    RecordDeclaration,
    FunctionDeclaration,
    TypeDeclaration,
    SimpleComponentDeclaration,
    RealDeclaration,
    IntegerDeclaration,
    BooleanDeclaration,
    StringDeclaration,
    InitialEquationAndAlgorithm,
    InitialEquation,
    InitialAlgorithm,
    EquationAndAlgorithm,
    Equation,
    Algorithm,
    ConnectorEquation
  );

  type Prefix = Enumeration(
    Any, None, Constant, Parameter,
    Discrete, Input, Output, Inner,
    Outer, Flow, Stream);

  Target target[:] = {Target.Any};
  Prefix prefix[:] = {Prefix.Any};
end AnnotationTarget;
```

One can now use such an annotation to restrict the applicability of the `Tuner` annotation:

```
record Tuner "Parameter to be optimized"
  import A = AnnotationTarget;
  parameter Boolean active = true;
  parameter Real min;
  parameter Real max;
  annotation(AnnotationTarget(
    target = {A.Target.RealDeclaration},
    prefix = {A.Prefix.Constant,
              A.Prefix.Parameter}
  ));
end Tuner;
```

This definition states, that the `Tuner` record is only to be used in an annotation on a `Real` declaration that has a constant or parameter prefix.

## 4.6 Summary

In this proposal we reuse and generalize existing concepts from the Modelica language. By doing so, we enable the handling of custom annotations. Let us recapitulate our proposed extensions to the Modelica language:

- Regular Modelica records can be used within annotations.
- Custom annotations can be applied in hierarchical modifiers.
- Hierarchical records are automatically supported in annotations.

For the moment, custom annotations are regarded as an additional feature but for the longer-term future an even extended concept shall be used to define also the standardized annotations. This would unify the language and reduce the complexity of the specification.

## 5 Using Meta-Information

In the previous section it was proposed how to store meta-information in a Modelica model. Due to its definition, it is not allowed to use this information in the model itself. The question is how a user or a tool can inquire the stored meta-information. Of course, if meta-information is related closely to a simulation model, most likely the respective Modelica translator has to extract and use the information in a tool specific way (for example meta-information related to the graphical representation of the model in the tool).

In this section, we analyze how to extract and utilize user-defined custom annotations for two possible applications: storing meta-information in FMI format and using meta-information in scripting. Of course many further applications are possible.

### 5.1 Storing Meta-Information in FMI Format

The Functional Mockup Interface (FMI) (*Blochowitz et al., 2011 and 2012*) is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML-files and compiled C-code. For details see, <https://www.fmi-standard.org/>. Most Modelica tools support the export and import of models in FMI format. The FMI standard stores all static model information in a file called `modelDescription.xml` in XML format. In particular, the information of all exposed variables are stored here, such as name, data type, unit, description text, variability, causality, etc. The FMI format 1.0 has been published in 2010 and

is supported by more as 40 tools. The release candidate of FMI 2.0 has been published in October 2013.

Since custom annotation variables are basically standard Modelica variables with all the attributes of Modelica variables, it is proposed to just store them as standard FMI variables and mark the “custom annotation” property in the name. In particular, the name of a custom annotation variable shall be:

```
<ComponentName>.annotation.<Custom
  AnnotationFullClassName>.<elementName>
```

Note, “annotation” is a reserved keyword in Modelica and therefore a name with “.annotation.” cannot be used as component name, so that a name clash between standard Modelica variable names and custom annotation variable names cannot occur.

As previously mentioned, via inheritance the same custom annotation can be used several times in a class annotation. This is handled by always defining a class annotation with a vector name where the index defines the inheritance order (the post-processing tool has then to define how to handle such vectors, e.g., to only use the first one, or utilize all definitions):

```
<ComponentName>.annotation[<i>].<Custom
  AnnotationFullClassName>.<elementName>
```

**Example:**

The custom annotation proposal has been partially implemented in a Dymola prototype for evaluation. The Modelica model

```
model Vehicle
  parameter Real p1=2 annotation(
    OptimSetup.Tuner(min=-2));
  ...
end Vehicle;

model ControlledDrive
  Vehicle car;
  ...
  annotation(OptimSetup.OptimOptions(
    tolerance=1e-3));
end ControlledDrive;
```

is stored in the following way in modelDescription.xml file with the Dymola prototype:

```
<?XML version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  fmiVersion="2.0"
  modelName="ControlledDrive"
  ...
>
...

<ModelVariables>
  <ScalarVariable
    name="car.p1"
    valueReference=" 16777216"
    description = "..."
```

```
causality      ="parameter"
variability    ="fixed">
<Real start   ="2" />
</ScalarVariable>

<ScalarVariable
  name="car.p1.annotation.
    OptimSetup.Tuner.min"
  valueReference="0"
  variability    ="constant">
  <Boolean start="true" />
</ScalarVariable>

<ScalarVariable
  name="annotation[1].OptimSetup.
    OptimOptions.tolerance"
  valueReference="0"
  variability    ="constant">
  <Real start="1.0e-3" />
</ScalarVariable>
...
</ModelVariables>
</fmiModelDescription>
```

The “annotation” in a name uniquely identifies the component to which this annotation is associated. For example “car.p1.annotation...” means that this variable is a custom annotation to variable “car.p1”. Usually, custom annotation variables are constants or parameters that are **evaluated** during translation and therefore these variables are stored with variability=“constant” and with a literal value in the xml file.

However, the above scheme is not restricted to this case: A custom annotation may contain time varying variables. In such a case the XML file alone is not sufficient to store the information, but a full FMU (Functional Mockup Unit) is needed, because the code to compute a time-varying variable at a particular time instant needs to be evaluated by the compiled C-code of the FMU.

If a tool already supports the export of a Modelica model in FMI format, then custom annotation variables have just to be included and stored in the standard variable tree.

**5.2 Using Meta-Information in Scripting Environments**

Typically, user-defined custom annotations are used to setup special analysis or synthesis environments, like optimization, nonlinear model predictive control, Monte Carlo simulation or uncertainty analysis. For this, the underlying model is needed, as well as the analysis-specific custom annotations defined in the model. If the custom annotations are stored in FMI format as proposed in the previous sub-section, the further processing is, in principal, simple: The information is stored in an XML-file and there are many scripting environments available, such as Java,

JavaScript, Matlab, Python, Ruby, to very easily read XML files and deduce the desired information from it. Therefore, meta-information about non time-varying custom annotations can be deduced in a straightforward way from all these scripting environments. If also time varying variables shall be supported, a scripting environment with FMI support is needed. Typically, the scripting environment will be used, in which the analysis or synthesis task can be directly formulated.

When using Dymola (*Dassault Systèmes, 2014*), there are scripts available to perform offline and online optimization, Monte Carlo simulation, calibration and others. It is natural to simplify the setup of these tasks by defining the model specific parts already in the respective model using custom annotations. Dymola uses the algorithmic part of Modelica as scripting language. Unfortunately, there is no API available to read XML files. This might also be not possible in a generic way, because the data structures supported by Modelica are not powerful enough for such applications.

For this reason, a special new Dymola API function was designed and implemented to read the variable information of an FMI 2.0 XML file (so the information about all exposed signals). The approach is demonstrated in the following code fragments:

```
function generateXXXsetup
  input String fileName;
protected
  ScalarVariable scalarVariable[:] =
    importScalarVariables(fileName);
algorithm
  ...
end generateXXXsetup
```

Function `generateXXXsetup` is a user-defined Modelica function to read an FMI XML file and generate the setup for the respective analysis task. The core is the new Dymola API function `importScalarVariables` that reads the `<ScalarVariable>` part of an FMI XML file and from this information all custom annotations can be deduced. The function returns a vector of records that has a complicated structure: Since Modelica does not have variant records, the different parts of the variable description are just appended. Some parts of the record definition are given below:

```
record ScalarVariable
  import Records.InternalRecords.*;
  import Records.Enumerations.*;
  String      name;
  Integer     valueReference;
  Causality   causality;
  Variability variability;
  Initial     initialDefinition;
  OptionalInteger previous;
```

```
  Type        variableType;
  RealAttributes realAttributes;
  IntegerAttributes integerAttributes;
  BooleanAttributes booleanAttributes;
  StringAttributes stringAttributes;
  IntegerAttributes
    enumerattionAttributes;
end ScalarVariable;

record RealAttributes
  OptionalString declaredType;
  OptionalString quantity;
  OptionalString unit;
  OptionalString displayUnit;
  OptionalReal   min;
  OptionalReal   max;
  Real           nominal;
  Boolean        unbounded;
  OptionalReal   start;
  OptionalInteger derivative;
  OptionalBoolean reinit;
end RealAttributes;

record OptionalReal
  Boolean present;
  Real   Value;
end OptionalReal;
...

```

Many attributes of `<ScalarVariable>` are optional. There is no special data type in Modelica to support optional values. For this reason, records “OptionalXXX” are used: Boolean element `present` defines whether the `Value` is defined, or was not given.

By inspecting all “`scalarVariable[i].name`” strings that have `.annotation.` in their name, the desired custom annotations can be deduced and can be utilized to generate the desired default setup of the respective environment.

## 6 Conclusions

There is a strong need to deal with meta-information in equation-based languages. We presented here a first design in order to enable a better handling of meta-information in Modelica: *custom annotations*.

These can be defined by the user and can be organized within packages. For the long term future, we hope to extend this concept to such a degree that a very high percentage of existing annotations can be covered by one unified concept and the specification can be simplified accordingly. For the near future, we are confident that the proposal will be the base for an enhancement of the Modelica language specification.

It is important to note that the presented design is a design proposal and by no means a definitive design. Also the Modelica Association offers a new process called Modelica Change Proposal (MCP) to

work a proposal into the language specification. The desired design for custom annotations will undergo this process and thereby being reviewed and eventually improved. The aim of this publication is hence to inform the public about the ongoing efforts for handling meta-information in Modelica and not to announce a definitive design decision.

## Acknowledgements

The custom annotation proposal in this article was developed within the ITEA2 project MODRIO. Partial financial support of the German BMBF and the Swedish VINNOVA for this development are highly appreciated.

This development was initiated by the MORIO project leader Daniel Bouskela, because associating meta-information with a Modelica model is a core feature needed in the MODRIO project.

Several variants of this proposal have been discussed in Modelica Design Meetings and via the Modelica trac system. Comments and improvement suggestions, especially from (alphabetical list): Volker Beuter, Peter Fritzson, Hans Olsson, Jesper Mattsson, Martin Sjölund, Michael Tiller, are highly appreciated.

A Dymola prototype to export custom annotations in an FMU was implemented by Hans Olsson and Karl Wernersson. Based on this prototype, Hans Olsson proposed to simplify the name lookup as it is described now in this paper.

The Dymola API function to read the variable description from an XML-file into Modelica was implemented in a prototype by Karl Wernersson. Based on this prototype, he proposed to refine the design, especially with respect to undefined attributes.

Custom annotations are already utilized in project MODRIO to develop optimization setups of various optimization problems. This work, carried out by Bernhard Bachmann, Martin Otter, Andreas Pfeiffer and Vitalij Ruge, gave valuable hints for the design of this custom annotation proposal.

## References

- Blochwitz T., Otter M., Arnold M., Bausch C., Clauß C., Elmqvist H., Junghanns A., Mauss J., Monteiro M., Neidhold T., Neumerkel D., Olsson H., Peetz J.-V., Wolf S. (2011): **The Functional Mockup Interface for Tool independent Exchange of Simulation Models**. Proceedings of the 8th International Modelica Conference, Dresden, March 20-22, pp. 105-114. Download: <http://www.ep.liu.se/ecp/063/013/ecp11063013.pdf>
- Blochwitz T., Otter M., Akesson J., Arnold M., Clauß C., Elmqvist H., Friedrich M., Junghanns A., Mauss J., Neumerkel D., Olsson H., Viel A. (2012): **Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models**. Proceedings of the 9th International Modelica Conference, September 3-5, Munich, pp. 173-184. Download: <http://www.ep.liu.se/ecp/076/017/ecp12076017.pdf>
- Burstall R., Strachey C. (2000): **Understanding Programming Languages**. Higher-Order and Symbolic Computation 13 :52.
- Coward D (2004). **JSR 175: A Metadata Facility for the JavaTM Programming Language**. Java Community Process. <https://www.jcp.org/en/jsr/detail?id=175#2> (Retrieved 2013-12-09).
- Dassault Systèmes (2014): **Dymola 2015 Alpha**. <http://www.Dymola.com>
- Modelica Association (2013): **The Modelica Language Specification, Version 3.3**. Download: <https://www.modelica.org/documents/ModelicaSpec33.pdf>.
- Zimmer D. (2008): **Multi-Aspect Modeling in Equation-Based Languages**. Simulation News Europe, Volume 18, No. 2, pp. 54-61
- Zimmer D. (2009): **An Application of Sol on Variable-Structure Systems with Higher Index**. 7th International Modelica Conference, Como, Italy.
- Zimmer D. (2012): **A Reference-Based Parameterization Scheme for Equation-Based Object-Oriented Modeling Languages**. 7th Vienna International Conference on Mathematical Modelling, Vienna, Austria.

# Modelica extensions for Multi-Mode DAE Systems

Hilding Elmqvist<sup>1</sup>, Sven Erik Mattsson<sup>1</sup>, Martin Otter<sup>2</sup>

<sup>1</sup>Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

<sup>2</sup>German Aerospace Center (DLR), Institute of System Dynamics and Control,  
D-82234 Wessling, Germany,

[Hilding.Elmqvist@3ds.com](mailto:Hilding.Elmqvist@3ds.com), [SvenErik.Mattsson@3ds.com](mailto:SvenErik.Mattsson@3ds.com), [Martin.Otter@dlr.de](mailto:Martin.Otter@dlr.de),

## Abstract

This paper describes a proposal for modeling systems with multiple operating modes, such as changing a controller from nominal operation to startup or shutdown or describing failure situations where the model structure is changing (e.g. an electrical line or a mechanical shaft breaks). This is achieved by extending the Modelica 3.3 synchronous state machines to continuous-time state machines having continuous-time models as “states”. Every model can be a “state” of a state machine, and in particular certain acausal models. Currently, no new language element is needed for Modelica, but a generalized semantics for state machines has to be introduced, such as “merge semantics for differential equations”. Symbolic transformations are still handled during translation, so the generated code is efficient and there is no run-time interpreter. On the other, this feature restricts the class of multi-mode systems that can be handled.

*Keywords: Multi-mode, failure simulation, dynamically changing states, continuous-time state machine, hybrid state machine.*

## 1 Introduction

The intention is to extend the scope of Modelica and model and simulate systems with multiple operating modes<sup>1</sup>. Examples:

- Changing a controller from nominal operation to startup, shutdown or manual operation.
- Structural changes of a physical model (e.g. modeling the separation mechanism of a two or three stage rocket).
- Describing failure situations where the model structure is changing (e.g. an electrical line or a mechanical shaft breaks).

- Describing failure situations where the model is completely changing (e.g. the normal behavior of a pump is a 0D model. In case of cavitation, a 1D model is needed to describe physics, requiring to switch dynamically from a 0D to a 1D model when cavitation occurs).

In general this means that the number of continuous-time states of the model might change dynamically during the simulation. Such models cannot be described with current Modelica, version 3.3 (*Modelica Association, 2012*), since in this case the basic requirement is that the number of continuous-time states of a model is fixed during a simulation.

The basic idea for multi-mode modeling is to extend the Modelica 3.3 synchronous state machines (*Elmqvist et. al., 2012*) to continuous-time state machines having continuous-time models as “states”. Every model can be a “state” of a state machine, including acausal models, such as a capacitor, or a rotational inertia. The number of continuous-time state variables can change at a transition of a state machine. No new Modelica language element is needed, but a generalized semantics for state machines has to be introduced, such as “merge semantics for differential equations”. The concepts have been evaluated with a Dymola prototype (*Dassault Systèmes, 2014*).

A related paper (*Bouissou et. al., 2014*) discusses the use of continuous-time state machines in Modelica for the modeling of stochastic hybrid systems by means of Piecewise Deterministic Markov Processes (PDMP). It focuses on how to handle the case when the transitions on continuous-time state machines depend on stochastic transitions.

Modeling state machines with differential equations in the states is a well-known approach in automata theory, called hybrid automata, see e.g. (*Henzinger 1996*). Such an approach is only of limited use when modeling physical systems, even if the Ordinary Differential Equations (ODEs) on a state are extended to Differential-Algebraic Equations (DAEs): The user has to enumerate all different configurations of a system and has to provide the equa-

---

<sup>1</sup> Section 1 and 2 of this article are an updated version of the internal report D4.1.2-M12 from the ITEA2 project MODRIO. The rest of this article is new material.

tions for every configuration and also provide all the details to switch between these configurations, see also next section.

MOSILAB (*Bastian et. al., 2010*) from Fraunhofer is an extension to Modelica by introducing continuous-time statecharts in Modelica and supporting DAEs on the states. This approach has similar drawbacks as hybrid automata, but is more powerful as hybrid automata due to the usage of Modelica.

Sol (*Zimmer, 2010*) is an experimental language to model variable structure systems with index changes. The large flexibility requires an interpreter for the simulation: Whenever the DAE index of a system is changed during simulation, the relevant equations of the model are newly symbolically processed and especially also newly index-reduced. The advantage is that a very large class of physical systems with variable structure can be described. The drawback is that an interpreter is needed at run-time, considerably reducing the simulation efficiency.

The approach presented in this paper introduces certain restrictions on what kind of changes can be made to the model at a mode change. This allows symbolic transformations still to be handled during translation, so the generated code is efficient and there is no run-time interpreter.

## 2 Continuous-time state machines with causal blocks

### 2.1 Synchronous state machines

In Modelica 3.3 (*Modelica Association, 2012*) synchronous state machines are defined. These state machines are only executed as sampled data systems. From the specification:

*Any Modelica block instance without continuous-time equations or algorithms can potentially be a state of a state machine. A cluster of instances which are coupled by transition statements makes a state machine. All parts of a state machine must have the same clock. All transitions leaving one state must have different priorities. One and only one instance in each state machine must be marked as initial by appearing in an initialState statement.*

### 2.2 Continuous-time state machines

In order to define multi-mode systems, the basic idea is to generalize the clocked state-machines from Modelica 3.3 to continuous-time. In a first step, two cases are distinguished:

1. All states and transitions of one state machine are clocked and belong to the same clock (= semantics in Modelica 3.3).
2. All states and transitions of one state machine are continuous-time (= new, additional semantics; discussed below).

In this section we only consider “causal” continuous-time systems, that is, the “states” must be “blocks” with defined “input” and “output” variables. As a result, every “state” block can be separately symbolically processed (such as BLT partitioning) assuming that all its inputs, `pre(...)` and arguments of `der(...)` are known and all other variables, especially outputs, `der(...)`, and arguments of `pre(...)` are unknown. Therefore, from a conceptual point of view a “state” is a set of ordinary differential equations (ODE) with known inputs  $\mathbf{u}$  and states  $\mathbf{x}$  and of explicit algebraic equations with unknown outputs  $\mathbf{y}$  computed in the block:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}, t)\end{aligned}$$

The outgoing transitions of the active state of a state machine are Boolean conditions that are continuously monitored and are transformed to event indicator signals (also called zero-crossing functions) that signal an event when the Boolean condition changes its value. At this time instant the numerical integration is stopped, the state machine switches to the new “state” and the ODE of this new state is either restarted or continues from the previous value of its continuous-time state variables when the “state” was active the last time.

As a simple example consider the continuous-time state machine in *Figure 1* with two “states”. This state machine consists of “states” that consist of completely unrelated blocks (in the upper “state” it is a drive train with clutches, and in the lower “state” it is a controlled electrical motor with load). At the start of the simulation the upper state is active and the drive train is simulated. At time = 1.5 s, the simulation of the drive train is stopped and the controlled electrical motor block starts simulating.

As can be seen, the number of continuous-time states changes dynamically during simulation and the state machine switches between unrelated models. There is the restriction that every “state” block needs to have a full initialization definition (e.g., starting from given start values of the continuous-time states, or starting from a steady-state condition which requires to solve a nonlinear algebraic equation system when the block becomes active the first time). When a “state” block is activated the first time, the initialization equations are first evaluated before the simu-



lation of the block is started. When a “state” block is re-entered, the block is re-initialized if the reset flag is set in the transition. Otherwise, the block continues from the values of the continuous-time variables state of the previous activation.

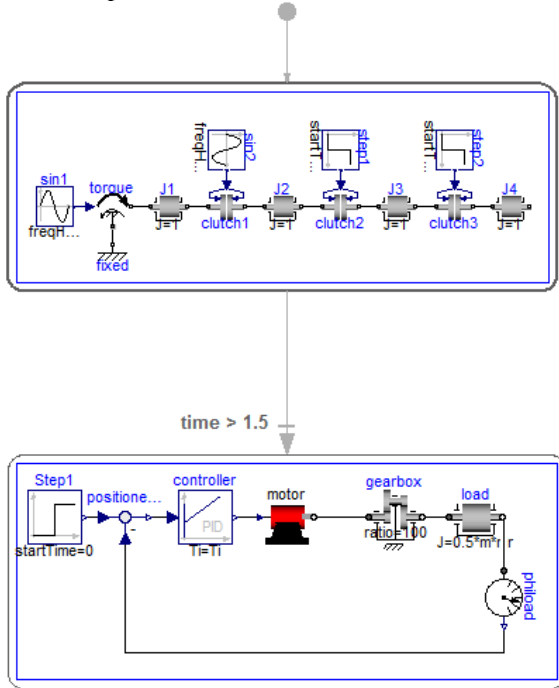


Figure 1: Continuous-time state machine

### 2.3 Hybrid automata

The system in Figure 1 is practically not very useful, since signal values are not exchanged between the different “state” blocks. More useful state machines are the already mentioned hybrid automata, see for example (Henzinger 1996). A simple example is shown in Figure 2 and Figure 3:

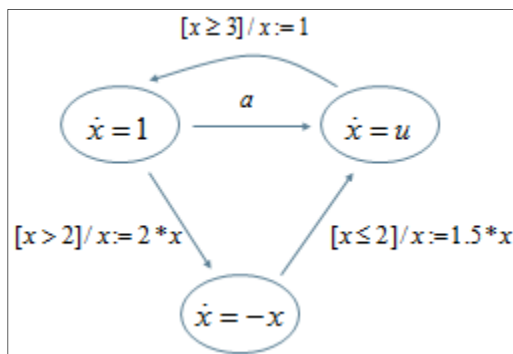


Figure 2: Hybrid automata with “a” as input signal.

On every “state” an ODE is present. The transition conditions consist of trigger conditions when to switch the state (e.g. “[ $x \geq 3$ ]”). Furthermore, at the transition it is defined in which way to reset the state of the ODE (e.g. “/ $x:=1$ ” means that continuous-time state  $x$  is reset to 1, before entering the target “state”). In the Dymola prototype, this state

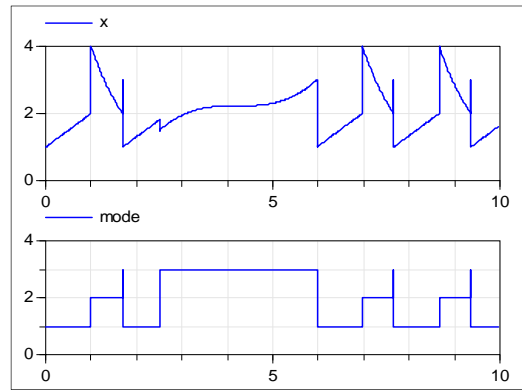


Figure 3: Solution of the hybrid automata from Figure 1 with  $a = \text{time} > 2.5$ ”.

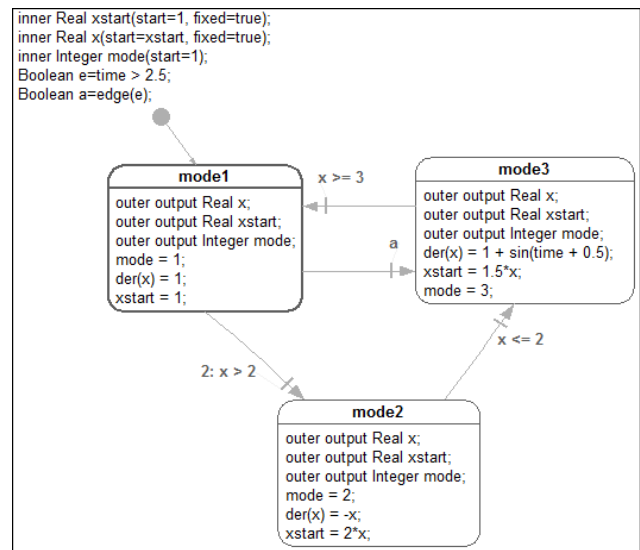


Figure 4: Hybrid automata from Figure 1 modeled in Modelica with indirect reset.

machine can be modeled with the proposed Modelica extension according to Figure 4. Here, every “state” is a block that contains the differential equation. Note, the same continuous-time state variable “ $x$ ” is used in all “states” (called “mode1”, “mode2”, “mode3” here), because this variable is defined with “outer output” in the mode blocks. Additionally, the start value  $xstart$  of the state is reported via inner/outer to all mode blocks. The expected semantics is that whenever entering a “state” and the reset flag in the transition is set, then the differential equation is newly initialized with the actual start value of “ $x$ ”.

The current semantics of the prototype implementation in Dymola is that when an immediate transition is used, the equations are activated once before the reset is made. This explains why  $xstart$  is modified in the destination state. The use of the construct:

```

inner Real xstart(start=1, fixed=true);
inner Real x(start=xstart, fixed=true);
    
```

is not standard Modelica since the variability of the start attribute is parametric. To allow any variability of the start expression would be one possible extension to Modelica to solve the re-initialization problem. However, the solution of using a “global” variable `xstart` is not elegant and is error prone. It would be better to allow such information for re-initialization to be associated with the transition. Further investigations are needed for designing the “right” extension.

Similarly as for clocked synchronous state machines, also for continuous-time state machines the “single assignment rule” holds for every variable. This means that variables in the different “states” must be merged. The merge-semantics for algebraic variables is identical to the merge-semantics of clocked synchronous state machines (see section 17.3.5 of the Modelica 3.3 specification). Variables that appear in the derivative operator, `der(..)`, are merged in the following, equivalent way:

```

der(x) := if activeState(state1) then
  expr1
elseif activeState(state2) then
  expr2
elseif
  ...
else last(der(x))
    
```

This means that in the example above, the equations for `der(x)` in the different “state” blocks are merged into the following single statement:

```

der(x) := if activeState(mode1) then
  1
elseif activeState(mode2) then
  -x
elseif activeState(mode3) then
  1 + sin(time+0.5)
else last(der(x))
    
```

### 3 Continuous-time state machines with acausal models

Hybrid automata are of limited use for physical system modeling, because the equations have to be first manually transformed into an input-output block and this is inconvenient and might be non-trivial. Furthermore, the graphical representation in an object diagram might be “not nice” due to input and output connections in a diagram that uses physical, that is acausal, connectors otherwise. Therefore, from a user point of view, it is important to support acausal models as “states”. In general this is non-trivial, because different “states” may require different symbolic handling of the equations in the environment since causality and the DAE index might change between “states” of a state machine. However, a quite

large class of acausal model “states” have been identified that can be reasonably handled.

As an example, consider the electrical circuit with two state machines in *Figure 6*. In the upper part of the electrical circuit a diode model is present as a “state”. At “`time > 0.9`” this state is left and state “`brokenDiode`” is activated consisting of a resistor with a large resistance. Note, that the “states” have “electrical pins” from which they are connected with the rest of the circuit. In the lower right part of the circuit diagram, another state machine is present. It consists of a capacitor “state” (modeled as a resistor in series with an ideal capacitor model).

At time = 0.3s, the hardware configuration is changed and the state switches to state “`R2`” consisting of a small resistance. At time 0.7s, the configuration is switched back. Note, that in this second case the number of continuous-time states is changing when switching.

In the rest of this section, the new method is described that allows us to handle such systems in an efficient way.

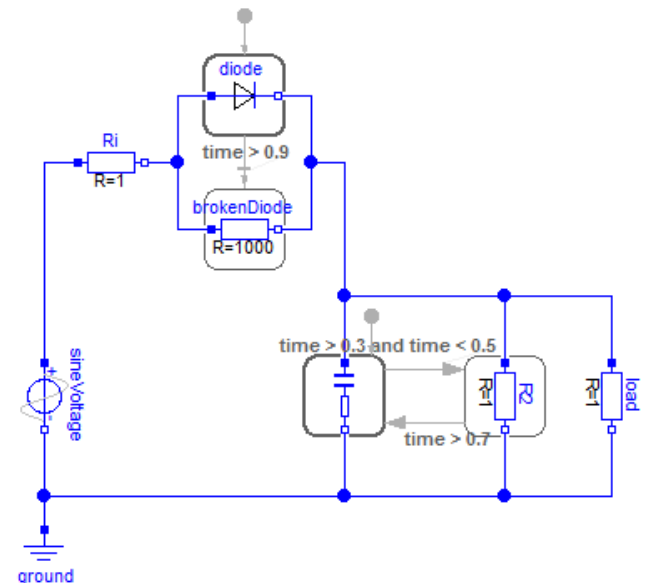


Figure 5: Circuit with two acausal state machines.

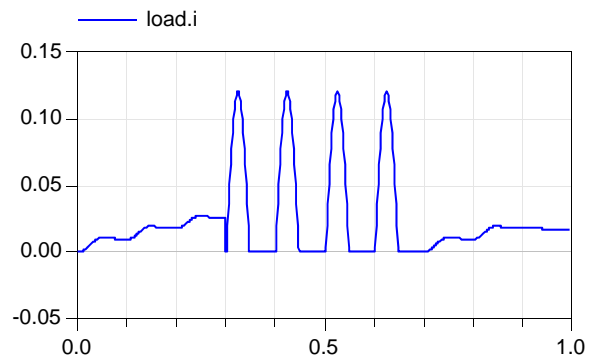


Figure 6: Simulation result of the circuit of Figure 5.

### 3.1 Basic idea for symbolic processing

The basic idea to symbolically process continuous-time state machines with physical connectors is explained at hand of the simple example in *Figure 7*. This circuit contains a state machine with two states, capacitor  $C$  (state 1) and resistor  $R_2$  (state 2). The simulation starts with capacitor  $C$  and after 0.5 s the state machine is switched to the resistor  $R_2$ . Conceptually, the same behavior can be achieved with the Modelica model shown in *Figure 8* where the models of the two previous two states are connected together with a special switch explained below.

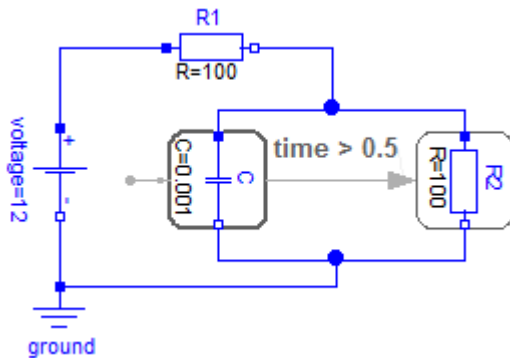


Figure 7: Simple circuit with an acausal state machine.

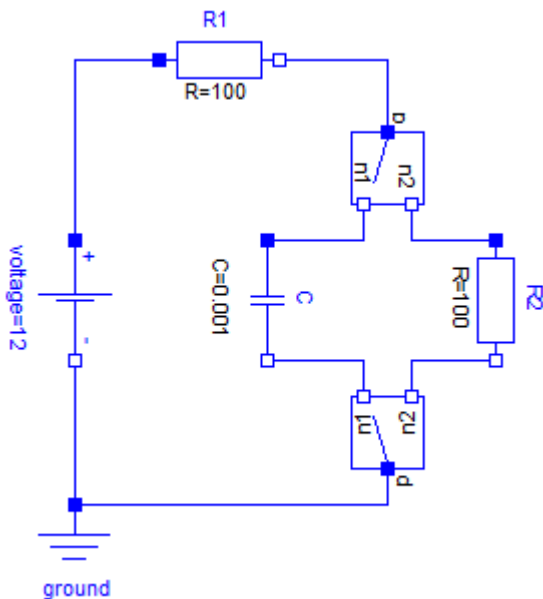


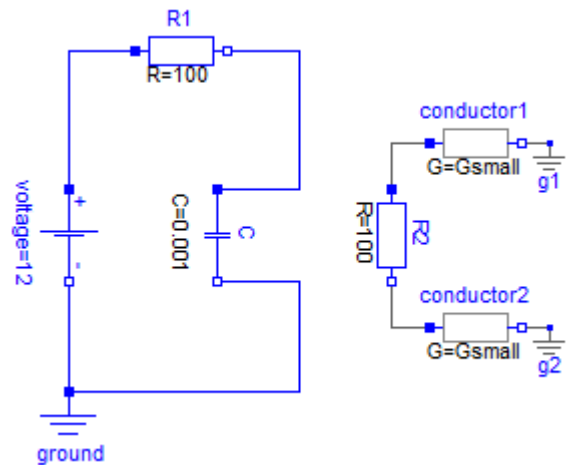
Figure 8: Simple circuit from Figure 7 implemented with special switches using standard Modelica.

Via the Integer input *state* to the switches it can be defined whether pin *n1* or pin *n2* is connected to pin *p*. In the circuit from above, it is defined as:

```
Integer state = if time <= 0.5 then 1
                else 2;
```

The effect of the two switch positions is demonstrated in *Figure 9*. If in state 1, the capacitor  $C$

Configuration if state = 1



Configuration if state = 2

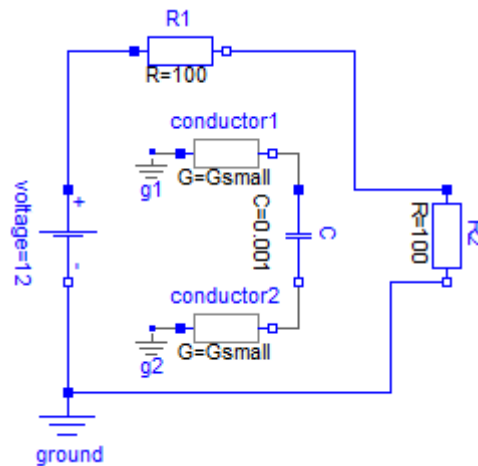


Figure 9: The two configurations of the simple circuit from Figure 8.

is connected with the rest of the circuit. The resistor  $R_2$  is connected with small (dummy) conductances to ground. This is necessary, because otherwise  $R_2$  would be “floating” and there would be missing equations for voltages. The currents into the pins would be zero since no environment would be attached. However, it is already stated in the resistor that the currents sum to zero. So there would be one equation too much for currents.

If in state 2, the resistor  $R_2$  is connected with the rest of the circuit. The capacitor  $C$  is connected with small (dummy) conductances to ground, similarly as the resistor in state 1.

It is clear that both configurations can be simulated with a standard Modelica simulator and that the variables of the “active” state will have identical results to the “active” state of the circuit in *Figure 7*. In order to achieve this behavior, the special switch models of *Figure 8* need to be defined as:

```

// Equation for potential variables
p.v = if state==1 then n1.v else n2.v;

// Equation for flow variables
0 = p.i + (if state==1 then n1.i
           else n2.i);

// Equation for not connected state
0 = if state==1 then Gsmall*n2.v-n2.i
    else Gsmall*n1.v-n1.i;

```

There are different variants to formulate the equations of this switch. The variant above has the advantage that one equation depends only on potential variables, but not on flow variables, and one equation depends only on flow variables and not on potential variables. This is the same principal structure as for every single configuration. For example, assume that in both configurations the potential variable equations need to be differentiated (due to a constraint between states). If in this case the combined potential variable equation of the switch is differentiated, then no flow variables are differentiated because the equation contains only potential variables.

The basic idea to handle acausal state machines can now be sketched:

- (1) Connect statements from the outside of a state machine to connectors on the states of a state machine are replaced by the equations of the special switch statement above.
- (2) All state machine states are removed and the equations of all states and all transitions are added to the rest of the model. The result is a standard DAE according to current Modelica.
- (3) The standard symbolic transformation algorithms are applied on this resulting DAE, such as Pandelides algorithm (*Pandelides 1988*), BLT, dummy derivative method (*Mattsson and Söderlind 1993*).
- (4) When generating code, all equations originally belonging to a state are de-activated (are not evaluated), when this state is not active. Furthermore, all continuous-time state variables from all non-active states are removed from the integrator<sup>2</sup>.

Note, applying rules (1) and (2) on the example from *Figure 7* results in the circuit of *Figure 8*. Due to rule (4), the code for the dummy conductors will not be executed and therefore the values of the conductances do not matter. The conductors are only important during the symbolic transformation phase

<sup>2</sup> A simple way to not integrate over deactivated states is to just report to the integrator that the derivatives of these states are zero, and otherwise keep the dimension of the state vector.

where structural properties of the equations are utilized.

Since the symbolic processing might differentiate equations and/or might lead to linear or non-linear systems of equations, this means, that also equations on states might be differentiated or algebraic systems of equations over states and non-states might be present.

In the rest of this paper, the above sketch is more formally defined, consequences and limitations are discussed and several examples are presented.

### 3.2 Mapping physical connections to equations

In this section it is defined how the `connect(...)` statement of Modelica is mapped to equations if one of the arguments is a connector on a state of a state machine.

#### 3.2.1 Connections between states of one state machine

##### Requirement 1:

It is not allowed to have a connection set where all connectors of the set belong to the same state machine and at least two connectors are on different states of this state machine.

Note, the states of a state machine are mutually exclusive. If there are connections between mutually exclusive states of a state machine, then the connection will never have an effect (because always only one connector will be active), and there will be missing equations.

#### 3.2.2 Connections between single potential-flow variable pairs

Assume a connector  $c_i$  present on a state  $i$  is defined by one potential variable  $p_i$  and one flow variable  $f_i$  and that  $n$  of these connectors from the same state machine are connected to connectors outside of this state machine.

A virtual connection node  $c_0$  is (conceptually) introduced outside of the state machine. All connections from  $c_i$  to connectors outside of the state machine are replaced by connections to  $c_0$  and connections from  $c_0$  to the original targets. As a result, the following connect statements will be present (the connect statements to the targets are handled according to current Modelica):

```

connect(c0, c1)
connect(c0, c2)
...
connect(c0, cn)

```

These connect statements are replaced by the following equations, where  $i$  characterizes the active state:

**Mapping Equations 1:**

$$\begin{aligned} \mathbf{p} &= [p_1, p_2, \dots, p_n]^T \\ \mathbf{f} &= [f_1, f_2, \dots, f_n]^T \\ i &= \text{activeState}() \end{aligned}$$

$$\begin{aligned} p_0 &= \mathbf{p}[i] \\ 0 &= f_0 + \mathbf{f}[i] \\ \text{for } j &= 1:n-1 \\ k &= \text{mod}(i+j-1, n) + 1 \\ 0 &= G_{small} \cdot \mathbf{p}[k] - \mathbf{f}[k] \end{aligned}$$

Note, the for-loop generates  $n - 1$  dummy equations describing a linear relationship between the potential and flow variables of the connectors on the states that are not active. Due to this mapping, the connect equations have the following structural dependency:

$$\begin{aligned} 0 &= g_1(p_0, \mathbf{p}, i) \\ 0 &= g_2(f_0, \mathbf{f}, i) \\ 0 &= \mathbf{g}_{3:n+1}(\mathbf{p}, \mathbf{f}, i) \end{aligned}$$

These are  $n + 1$  equations for  $n + 1$  connectors, where

- one equation depends on all potential variables of the connection set and on the active state,
- one equations depends on all flow variables of the connection set and on the active state,
- $n - 1$  equations depend on all potential and all flow variables of the connectors that are on the state machine and on the active state.

The equations  $\mathbf{g}_{3:n+1}$  are in principal only relevant for the symbolic analysis. During simulation, these equations are never evaluated, because they are deactivated for the active state.

If these equations appear in an algebraic loop, a simple implementation might just compute the solution without taking into account that the equations are deactivated and then ignore the computed value for the deactivated variables. In this case the value of  $G_{small}$  matters because the value is used during the solution of the equation system, and if the value becomes too small, the Jacobian of the equation system may become badly conditioned. In this case  $G_{small}$  should be in the order of the other elements of the equation Jacobian (or if this information is not known,  $G_{small} = 1$  might be used).

An efficient implementation requires to rewrite algebraic equation systems when a new state becomes active. For example, assume a linear system of equations is present during simulation and for state  $i = 1$  it has the form:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{0}^T & G_{small} & -1 & \mathbf{0}^T \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} & \mathbf{A}_{34} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{z}_1 \\ p_2 \\ f_2 \\ \mathbf{z}_3 \end{bmatrix} = \begin{bmatrix} \dots \\ \dots \\ \dots \end{bmatrix}$$

Since the variables and equations of state  $i = 2$ , especially  $p_2, f_2$ , are deactivated (and the variables hold their values) if this state is not active, this linear system of equation can be simplified for state  $i = 1$  to:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{14} \\ \mathbf{A}_{31} & \mathbf{A}_{34} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_3 \end{bmatrix} = \begin{bmatrix} \dots \\ \dots \end{bmatrix}$$

With this type of equation handling,  $\mathbf{g}_{3:n+1}$  are never evaluated during simulation and the value for  $G_{small}$  does not matter.

Note, when implementing a generic switch for  $n$  connectors, the ‘‘Mapping Equations 1’’ can be compactly defined in Modelica:

```
input Integer state;
parameter Integer nStates;
Pin p, n[nStates]
equation
  p.v = n[state].v;
  0 = p.i + n[state].i;
  zeros(nStates-1) = {Gsmall*
    n[mod(state+i-1, nStates)+1].v -
    n[mod(state+i-1, nStates)+1].i
    for i in 1:nStates-1};
```

### 3.2.3 Connections between input-output connectors

Assume state  $i$  of a state machine is an input-output block with an input  $u_i$  and an output  $y_i$  and that  $n$  of these connectors from the same state machine are connected to connectors outside of this state machine – from a node  $u_0$  to the inputs  $u_i$  and from the outputs  $y_i$  to a node  $y_0$ . This means that the following connect statements are present:

```
connect(u_0, u_1)      connect(y_1, y_0)
connect(u_0, u_2)      connect(y_2, y_0)
...
connect(u_0, u_n)      connect(y_n, y_0)
```

These connect statements are replaced by the following equations, where  $i$  characterizes the active state:

**Mapping Equations 2:**

$$\begin{aligned} \mathbf{u} &= [u_1, u_2, \dots, u_n]^T \\ \mathbf{y} &= [y_1, y_2, \dots, y_n]^T \\ \mathbf{u} &= u_0 \cdot \text{ones}(n) \\ y_0 &= \mathbf{y}[\text{activeState}()] \end{aligned}$$

These are  $n + 1$  equations for  $n + 1$  connections. When a state  $j$  is not active, an arbitrary value can be provided for its input  $u_j$ . For simplicity, just the

overall input  $u_0$  is provided, so  $u_j = u_0$ . Again, because equations on not active states are deactivated during simulation, these equations are only present during the symbolic transformation, but not during run-time. Therefore, the actually provided value for the input does not matter. Additionally, to the mapping rules above, the standard input-output semantic restrictions (e.g. an output can be only connected to one input), must be relaxed, so that this rule holds only for the active state, but not for deactivated states.

In Figure 10 an example with two input-output blocks is given: The simulation starts with block `firstOrder`, a first order block. When  $time > 0.5$ , the state machine switches to block `secondOrder`. Entering `secondOrder` re-initializes the block to the output of state `firstOrder` (more details on re-initialization are given in section 3.3).

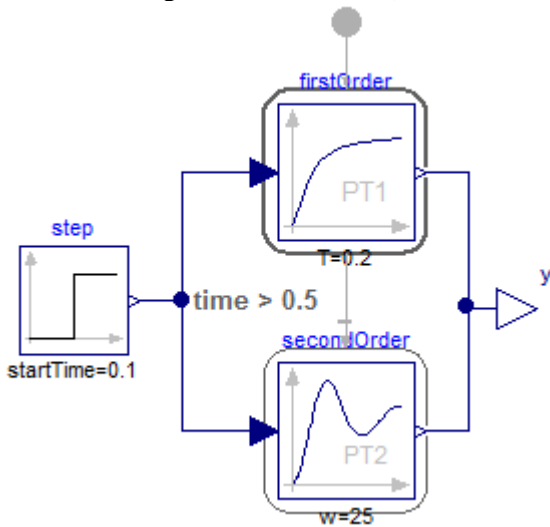


Figure 10: State machine switching between two input-output blocks.

Simulation results are shown in Figure 11. Here a first order behavior is seen for the first 0.5 s. Afterwards a second order behavior occurs.

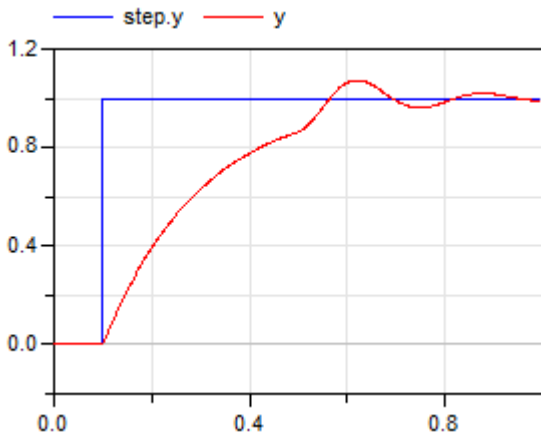


Figure 11: Simulation results of the state machine of Figure 10.

### 3.2.4 Equation rewriting to enhance efficiency

The approach from section 3.2.2 might lead to algebraic equation systems where the size of the systems is unnecessarily large. In some cases these sizes might be reduced with the following rewriting rules for connect equations:

First, run the Pantelides algorithm (Pantelides 1988) and perform the BLT (Block Lower Triangular) transformation on the differentiated problem.

Second, if the following equations of one connection set (see “Mapping Equations 1”),

$$\begin{aligned} p_0 &= \mathbf{p}[i] \\ 0 &= G_{small} \cdot \mathbf{p}[k] - \mathbf{f}[k] \end{aligned}$$

or their differentiated form, appear in an algebraic equation system having all potential variables  $\mathbf{p}[k]$  as unknowns, but not the flow variables  $\mathbf{f}[k]$ <sup>3</sup>, then the equations above can be reformulated to:

```
for i = 1:n
  p[i] = if activeState(i) then p_0 else last(p[i])
```

where  $\mathbf{last}(\mathbf{p}[i])$  is the value of  $\mathbf{p}[i]$  when state  $i$  was active the last time (so it is a known value).

The proof for this rewriting is given for  $n = 2$ . For  $n > 2$ , similar arguments can be given. So, assume  $n = 2$ . Then, the equations from the connect-statements have the following form for the different states, according to “Mapping Equations 1”:

```
if activeState(1) then
  p_0 = p_1
  0 = f_0 + f_1
  0 = G_small * p_2 - f_2
else
  p_0 = p_2
  0 = f_0 + f_2
  0 = G_small * p_1 - f_1
end if
```

Since it is assumed that the flow variables  $f_1$  and  $f_2$  are known (are computed somewhere else), the dummy conduction equations can be solved for the potential variables (e.g.  $p_2 := f_2 / G_{small}$ ). This means that for the non-active states, the potential variables are known. Instead of computing them by a dummy conduction equation, alternatively another known value can be used (because all equations of non-active states are anyway de-activated) and especially  $\mathbf{last}(\mathbf{p}[i])$ . As a result, the equations above are equivalent to:

<sup>3</sup> The flow variables do not appear in the algebraic loop if all are continuous-time states or are computed from continuous-time states.

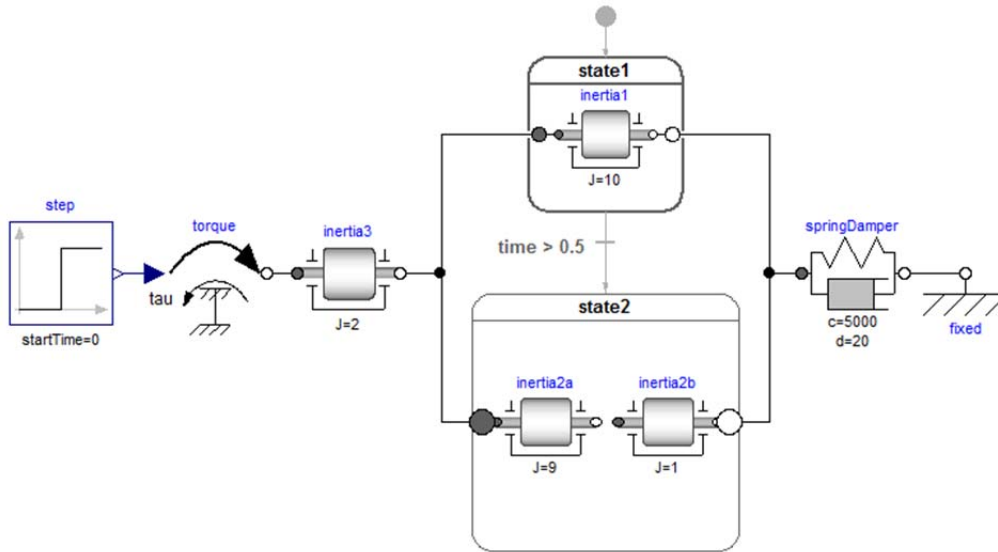


Figure 12: Breaking inertia that requires index reduction and algebraic loop handling over a state machine, as well as re-initialization when entering state2.

```

if activeState(1) then
   $p_0 = p_1$ 
   $0 = f_0 + f_1$ 
   $p_2 = \text{last}(p_2)$ 
else
   $p_0 = p_2$ 
   $0 = f_0 + f_2$ 
   $p_1 = \text{last}(p_1)$ 
end if

```

This equation set is in turn equivalent to:

```

 $p_1 = \text{if activeState(1) then } p_0 \text{ else last}(p_1)$ 
 $p_2 = \text{if activeState(2) then } p_0 \text{ else last}(p_2)$ 
 $0 = f_0 + \text{if activeState(1) then } f_1 \text{ else } f_2$ 

```

and the proof is complete.

After re-writing the equations, all symbolic algorithms are re-run. Usually, the equation systems will become smaller, since the equations depend now only on two variables, and not on many variables. Intuitively this rewriting means that  $p_0$  is computed outside of the state machine and propagated to the states of the state machine. Therefore, the previous algebraic loop over all the states is broken. A prerequisite for this rewriting is that for all non-active states it is sufficient to treat the potential variables as inputs. A sufficient condition for this to be possible is that the flow variables have been already computed somewhere else.

In a similar way, the equations can be rewritten, if the flow variable equation and the dummy conductor equations are in an algebraic loop.

### 3.3 Re-Initialization

When changing from one state to another one, the DAE of the target state must usually be initialized. This is basically performed with the methods from section 2.3 as demonstrated at hand of the example in Figure 12. This example models a drive train, where a rotational inertia breaks during the operation. In particular, this models consists of

- a state1 with inertia1 and J=10,
- a state2 with inertia2a (J=9) and inertia2b (J=1) that are not connected,
- an inertia3 outside of the state machine that is connected to inertia1 and inertia2a and is driven by a step torque<sup>4</sup>, and
- a spring-damper that is connected to inertia1 and inertia2b.

At time = 0.5, the state machine switches from state1 to state2 and therefore inertia1 is replaced by two unconnected inertias that have together the same moment of inertia as inertia1. In other words, the “breaking” of inertia1 is modelled. Note, that the number of continuous-time states is changing (there are 2 continuous-time state when in state1 and 4 when in state2).

With the generation of connect equations in section 3.2 and the sketched symbolic processing of the overall DAE, this system gives rise to index reduction between inertia3, inertia1 and inertia2a

<sup>4</sup> inertia3 is only present to demonstrate index reduction and algebraic loops over a state machine

and due to this index reduction an algebraic loop between equations of these components are present. The Dymola prototype handles this correctly.

When switching from `state1` to `state2`, `inertia1` has some angle and angular velocity and the inertias on `state2` need to be appropriately initialized. Since `inertia2a` is rigidly attached to `inertia3`, no initialization of `inertia2a` is needed. However, the continuous-time states of `inertia2b` need to be initialized to the states of `inertia3`. This is performed in the following way:

```
inner Real phi(start=0, fixed=true) =
    inertia3.phi;
inner Real w(start=0, fixed=true) =
    inertia3.w;
...
// in state2
import R=Modelica.Mechanics.Rotational;
outer Real phi;
outer Real w;
R.Components.Inertia inertia2b(J=1,
    phi(start=phi, fixed=true),
    w(start=w, fixed=true));
```

On the top level the inner variables `phi` and `w` are associated with the corresponding variables of `inertia3`. In `state2` these variables are declared as `outer` and used as `start` values for `inertia2b`. The semantics is that when `state2` is entered, then the variables of `state2` are re-initialized to their start values. A result plot of the angular velocities of the inertias on the two states is shown in *Figure 13*.

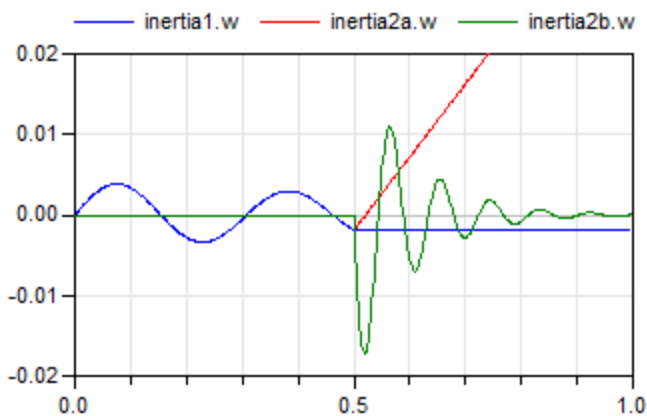


Figure 13: Simulation result for the breaking inertia of Figure 12.

As can be seen, the amplitude and frequency of `inertia2b` increases with respect to `inertia1`, because its moment of inertia is much smaller.

### 3.4 Limitations

The question arises which types of models cannot be handled with the proposed approach? First, independently of the symbolic algorithms used, the map-

ping of connect statements to equations as defined in section 3.2 is always correct. It is clear that there must be limitations when applying the standard symbolic algorithms on the resulting set of equations, because the structure of the equations depends on the active state and this dependency is not taken into account by the standard algorithms. For example assume that the following equation is present

$$p.v = \text{if } \text{state}==1 \text{ then } n1.v \text{ else } n2.v$$

when mapping some connect statements to equations. Assume that `p.v` and `n1.v` are states, but `n2.v` is not. Then this state constraint is not detected and `n2.v` is always computed from `p.v` and from `n1`. Of course, this will fail (will give a division by zero) in state 1. The correct handling would be that in state 1 the equation `p.v = n1.v` is present and if both variables are states, this equation must be differentiated. However, the standard algorithms do not take this into account and it seems also non-trivial to generalize.

In *Figure 14* there is a state machine with a capacitor `C1` and a resistor `R2`. These two states are connected in parallel to a capacitor `C2`. This model cannot be handled with the proposed method (and will give a run-time error that a matrix is singular), because in the capacitor state there is a state variable constraint between `C1` and `C2` and in the resistor state there is no such state constraint.

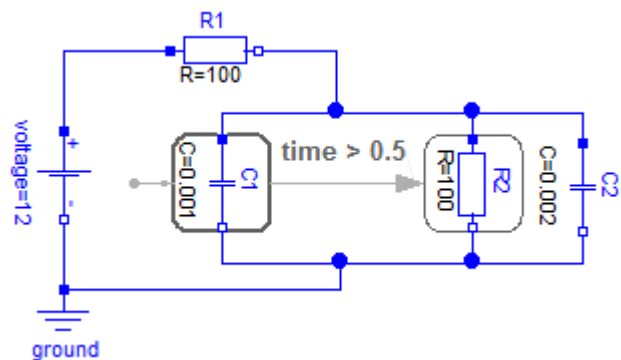


Figure 14: Parallel capacitors that cannot be handled due to different state constraints in the different states.

In *Figure 15* there is a state machine with a connection of two flanges in `state1` and no connection in `state2` (this is defined by connecting zero-torques to the two flanges). This state machine is placed between two inertias. The model describes a breaking inertia in a more natural formulation as in *Figure 12*. This model cannot be handled with the proposed method (and will give a run-time error that a matrix is singular), because in `state1` there is a state constraint between `inertia1` and `inertia2` and in `state2` there is no such state constraint.



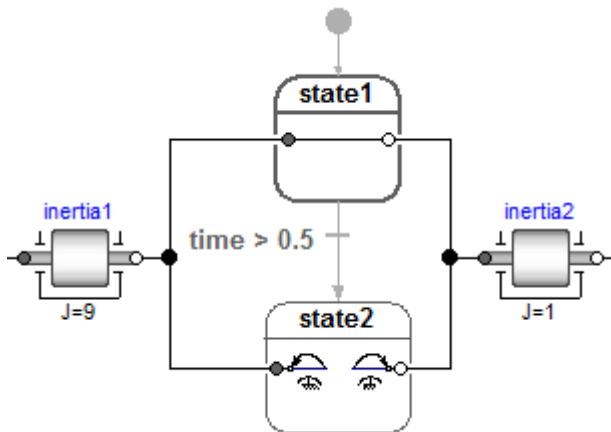


Figure 15: Breaking inertia that cannot be handled due to different state constraints in the different states.

It is not nice that such models lead to an error only when simulating the model. It is also difficult to deduce the source of the problem from an error message stating that a matrix is singular. In principal, the diagnostics can be improved, so that such errors occur during translation with an understandable error message:

The sorted equations are inspected and every algebraic equation system that depends on a state of a state machine is processed again: The equations of an algebraic equation system are partitioned (so the assignment algorithm is applied) for every state machine state taking into account the equation structure of the particular state only. If one of the assignments fails, the equation system is structurally singular for the selected state and therefore the model cannot be handled.

## 4 Conclusions

A proposal is presented for modeling variable structure systems with dynamically changing number of states in Modelica by extending the synchronous clocked state machines to continuous-time state machines. With this extension it is straightforward to model hybrid automata. However, hybrid automata are not practical to use for physical system modeling. A novel extension is proposed to use acausal models as states of a state machine. By mapping connections to connectors on a state machine in a particular way on equations, the standard symbolic processing for Modelica models can be applied. This approach allows already handling a large class of useful variable structure systems with dynamically changing sizes of continuous-time states.

Models cannot be handled with this new method, if connections between state and non-state components lead to constraints on continuous-time state

variables that vary for the different state machine states.

The proposal is not yet complete. Especially, mappings for all connector types of Modelica need to be still defined, especially for multi-body and for fluid systems. Additionally, the switching between DAEs may lead to Dirac impulses, if not properly re-initialized (or it must be modelled in a way that impulses occur, due to the underlying approximation of the reality). Furthermore, algebraic equation systems over states need to be analyzed in more detail, especially in combination with the dummy derivative method. It is planned to work on these topics in the near future.

## Acknowledgements

This paper is based on research performed within the ITEA2 project MODRIO. Partial financial support of the Swedish VINNOVA and the German BMBF for this development are highly appreciated.

## References

- Bastian J., Clauß C., Enge-Rosenblatt O., and Schneider P. (2010): **MOSILAB – a Modelica solver for multi-physics problems with structural variability**. 1st Conference on Multiphysics Simulation - Advanced Methods for Industrial Engineering 2010. Download: <http://publica.fraunhofer.de/starweb/servlet.starweb?path=urn.web&search=urn:nbn:de:0011-n-1355711>
- Dassault Systèmes (2014): **Dymola 2015 Alpha**. <http://www.Dymola.com>
- Elmqvist H., Gaucher F., Mattsson S.E., Dupont F. (2012): **State Machines in Modelica**. Modelica'2012 Conference, Munich, Germany, Sept. 3-5, 2012. Download: <http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf>
- Bouissou M., Elmqvist H., Otter M., and Benveniste A. (2014): **Efficient Monte Carlo simulation of stochastic hybrid systems**. Modelica'2014 Conference, Lund, Sweden, March 10-12.
- Henzinger T.A. (1996): **The Theory of Hybrid Automata**. Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), pp. 278-292.
- Mattsson, S.E. and G. Söderlind (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14 pp. 677-692.
- Modelica Association (2012): **The Modelica Language Specification, Version 3.3**. Download: <https://www.modelica.org/documents/ModelicaSpec33.pdf>.
- Pantelides C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, 9(2), pp. 213-231.
- Zimmer D. (2010): **Equation-Based Modeling of Variable-Structure Systems**. Dissertation, ETH Zürich, No. 18924. Download: [http://www.inf.ethz.ch/personal/fcellier/PhD/zimmer\\_phd.pdf](http://www.inf.ethz.ch/personal/fcellier/PhD/zimmer_phd.pdf)



# Integrated Debugging of Equation-Based Models

Martin Sjölund<sup>1</sup>, Francesco Casella<sup>2</sup>, Adrian Pop<sup>1</sup>, Adeel Asghar<sup>1</sup>, Peter Fritzson<sup>1</sup>,  
Willi Braun<sup>3</sup>, Lennart Ochel<sup>3</sup>, Bernhard Bachmann<sup>3</sup>

<sup>1</sup>Programming Environments Laboratory

Department of Computer and Information Science

Linköping University, Linköping, Sweden

<sup>2</sup>Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy

<sup>3</sup>Dept. Mathematics and Engineering, University of Applied Sciences,

D-33609 Bielefeld, Germany

{adrian.pop,martin.sjolund,adeel.asghar,peter.fritzson}@liu.se

francesco.casella@elet.polimi.it, {bernhard.bachmann,lennart.ochel,willi.braun}@fh-bielefeld.de

## Abstract

The high abstraction level of equation-based object-oriented languages (EOO) such as Modelica has the drawback that programming and modeling errors are often hard to find. In this paper we present the first integrated debugger for equation-based languages like Modelica, which can combine static and dynamic methods for run-time debugging of equation-based Modelica models during simulations. This builds on and extends previous results from a transformational static equation debugger and a dynamic debugger for the algorithmic subset of Modelica.

*Keywords: Modelica, Debugging, Modeling and Simulation, Transformations, Equations, Algorithmic Code, Runtime Errors, Tracing, Solver Failures*

## 1 Introduction

The advanced development of today's complex products requires integrated environments and equation-based object-oriented declarative (EOO) languages such as Modelica [10][14] for modeling and simulation.

The increased ease of use, the high abstraction, and the expressivity of such languages are very attractive properties. However, the downside of this high-level approach is that understanding the root causes of unexpected behavior and numerical errors of simulation model is very difficult, in particular for users who are not experts in simulation methods.

The main reason of this difficulty the fact that lots of sophisticated symbolic and numerical transformations are applied to the original model in order to eventually obtain the executable simulation code, in which errors and problems do occur. An effective debugging environment should then guide the end user

back and forth through the numerical results and all the performed symbolic transformations of the model, in order to quickly find and correct the causes of errors. This paper presents the integrated debugger of the OpenModelica tool suite, including a graphical user interface integrated with the OpenModelica Connection Editor (OMEdit) GUI. This builds on and extends previous results from a transformational static equation debugger [6][7] and a dynamic debugger [1][3][4] for the algorithmic subset of Modelica.

Despite the fact that debugging environments have been the subject of extensive research and implementation work in the field of computer science, to the best of the authors' knowledge this is the first documented operational debugging environment for equation-based modeling languages supporting dynamic debugging of equation-based mathematical models as well as algorithmic code in an integrated way.

The rest of the paper is structured as follows: The debugging procedure is outlined in Section 2 and the GUI in Section 3. The tracing of equation transformation is discussed in Section 4, while Section 5 discusses the issues of interfacing with the run-time simulation executable. In Section 6, some example models are shown, illustrating how the debugger can help their troubleshooting. Section 7 discusses background and related work, Section 8 states the current implementation status at the time of this writing, and Section 9 concludes the paper.

## 2 Overall Debugging Procedure

The debugger should support three basic scenarios:

- The simulation stops at a certain time step, or during initialization, because of a numerical runtime error;

- A complete simulation run has been performed successfully, but some variables exhibit suspicious or clearly wrong values;
- A breakpoint is inserted to stop the integration either at a certain given value of the time variable, or when some user-supplied condition is triggered. In this case, it should be possible to restart the simulation (and possibly to set a new breakpoint)

The different functionalities of the debugger are specified in more detail in the following sub-sections.

## 2.1 Types of Debugging Activities

We divide the problem of debugging the execution (i.e., the numerical simulation) of an equation-based model into three different areas:

- *Initialization.* Before starting the simulation, consistent initial conditions are computed by solving a set of initial equations. In the following, it is assumed that this is done by using multiple optimization strategies, such as alias variable elimination, BLT partitioning, tearing, etc.
- *Causalization.* It is also assumed that the solution of the differential-algebraic equations over time is obtained by a two-stage strategy. In the casualization stage, the DAEs are solved for the derivatives by using multiple optimization strategies, such as symbolic index reduction as well as the ones previously mentioned.
- *Time integration.* The computed derivatives (and possibly their Jacobian matrix) are then passed to ODE solvers, such as DASSL, Runge-Kutta, Radau, etc., that advance the solution of the system over time

## 2.2 Debugging Initialization and Causalization Problems

For the purpose of debugging, initialization and causalization share a common structure despite using different numerical solvers. They can be represented using a similar GUI. The only difference is that the set of equations and unknowns for initialization is larger than for causalization, as it also includes the state variables and the parameters, as well as the initial equations and parameter-binding equations. Also, the simulation code to solve both problems is usually generated by the Modelica tool itself, so it is fairly straightforward for the tool developers to add all kind of instrumentation to it for debugging purposes.

Variables are matched to the equations that are used to solve them. If an error has occurred while trying to compute a certain variable or a certain set of variables

for strong components in the BLT, the error (e.g., division by zero, logarithm of a negative number, singular linear system of equations, etc.) is reported in the context of the equation as it has been transformed in order to solve it efficiently at run time. Then, it is possible to backtrack step-by-step each stage of the transformations of each equation, up to the original equations in the source code.

This activity can also be carried out in the absence of errors, either when a breakpoint is triggered, or when the values at a specific time step are inspected after the simulation run has been performed. Assuming that some variable(s) have suspicious, or maybe clearly wrong values, one starts analyzing the equations that were used to compute them, going backwards in the causality chain determined in the BLT, and trying to locate the model error that caused the computation of the wrong values.

The solution of the equation(s) also depends on the values taken by all the other known variables showing up in the equations, either states or other unknown variables previously computed in the BLT. The debugger allows to inspect the values taken by these variables, as well as the equation(s) in which they were solved for. Then, the same activities will be possible recursively on this new set of equations: understanding where they come from in the equation transformation chain, as well as inspecting the values of the variable(s) they depend upon.

## 2.3 Debugging Time Integration Problems

The requirements for the debugging of *time integration* problems are quite different. Unrecoverable errors generated by the ODE solver should be reported to the debugger using some kind of unified representation (e.g., using XML), which is as independent as possible from the specific solver used. Of course, some errors will only make sense for a subset of solvers; for example, singular Jacobians are only relevant in the case of implicit solvers; event chattering is only relevant for solver with state event detection.

The first kind of error that can arise in solvers with state event detection based on zero crossing function is chattering: if a large number of events takes place in a very short time interval, then the debugger reports the corresponding zero-crossing functions and allows to back-track them to their original formulation in the source code, as well to inspecting the values of all the variables involved in them in the last accepted time steps.

It may also be the case that chattering arises without any event being generated, if the `noEvent()` operator is incorrectly placed around a discontinuous expression

inside a model equation, or if some functions in the model generate results which are discontinuous w.r.t. their inputs (recall that Modelica functions do not generate events). This situation can be detected by monitoring the step size, and detecting the fact that the step size has been reduced to very small values for a very large number of step sizes.

In order to identify the root cause of the problem, it is necessary that the ODE solver can report which component(s) of the state vectors have the largest estimated errors, and are thus mainly responsible for the excessive step size reduction. The debugger will then point the end user to the equations that are used to compute the corresponding derivatives, using the same mechanism adopted for the initialization and casualization steps. Wildly oscillating values of the derivatives will be observed across the last time steps, and it will then be possible to analyze the expressions leading to these oscillations, eventually locating the root cause of the problem.

Another possible error can arise at the ODE solver level if the underlying differential equations have a finite escape time, i.e., one or more elements of the state vector go to infinity as time approaches a certain finite value. The main symptom in this case is very similar to the previous case, i.e., the step size is greatly reduced and the simulation seems stuck at a certain point in time.

The root cause can also be identified in this case if the solver reports the component(s) of the state record that mostly contribute to the error estimate, so that the debugger can allow the user to inspect the equation(s) that compute the corresponding derivatives. The values of these derivatives will constantly grow from one step to the next one, rather than oscillating wildly as in the previous case. Again, by careful inspection and analysis, it might be possible to understand the root cause of the problem and fix it.

## 2.4 Debugging Homotopy-based Initialization Problems

If the `homotopy()` operator is used for initialization, two extra stages are added to the debugging of the initialization problem. First, the set of initial equations using the simplified expression is presented. The BLT structure of this problem might be substantially different (and hopefully simpler) than that of the actual initialization problem, but the way it is presented in the GUI to the user for analysis is the same as for the actual initialization problem.

The second stage is the homotopy transformation. From a GUI perspective, this is very similar to the simulation phase as there are several steps involved. Each might be accepted, rejected, or eventually fail if the errors cannot be recovered by taking shorter steps. Also, similarly to the simulation phase, errors might be reported that arise while solving the equations in the BLT sequence (as in the initialization and casualization problems), but also some system-level errors might be reported by the homotopy solver itself, e.g., in case of homotopy path bifurcations, similarly to problems reported by the ODE solver during time integration.

The GUI is therefore similar to the one used for debugging errors during simulation, with the following differences:

- The set of unknowns includes states and parameters;
- the set of equations include initial equations and parameter-binding equations
- All occurrences of the homotopy operator [14] in the equations are transformed into  $\lambda * actual\_expr + (1 - \lambda) * simplified\_expr$ ;
- The independent variable which is stepped is not time but rather the  $\lambda$  homotopy parameter.

## 3 Debugger Graphical User Interface

In order to visualize the transformations performed and the operations taken by the solver to solve for a variable and its corresponding equation(s), a *transformations browser* (Figure 1; Figure 2; Figure 3) has been created.

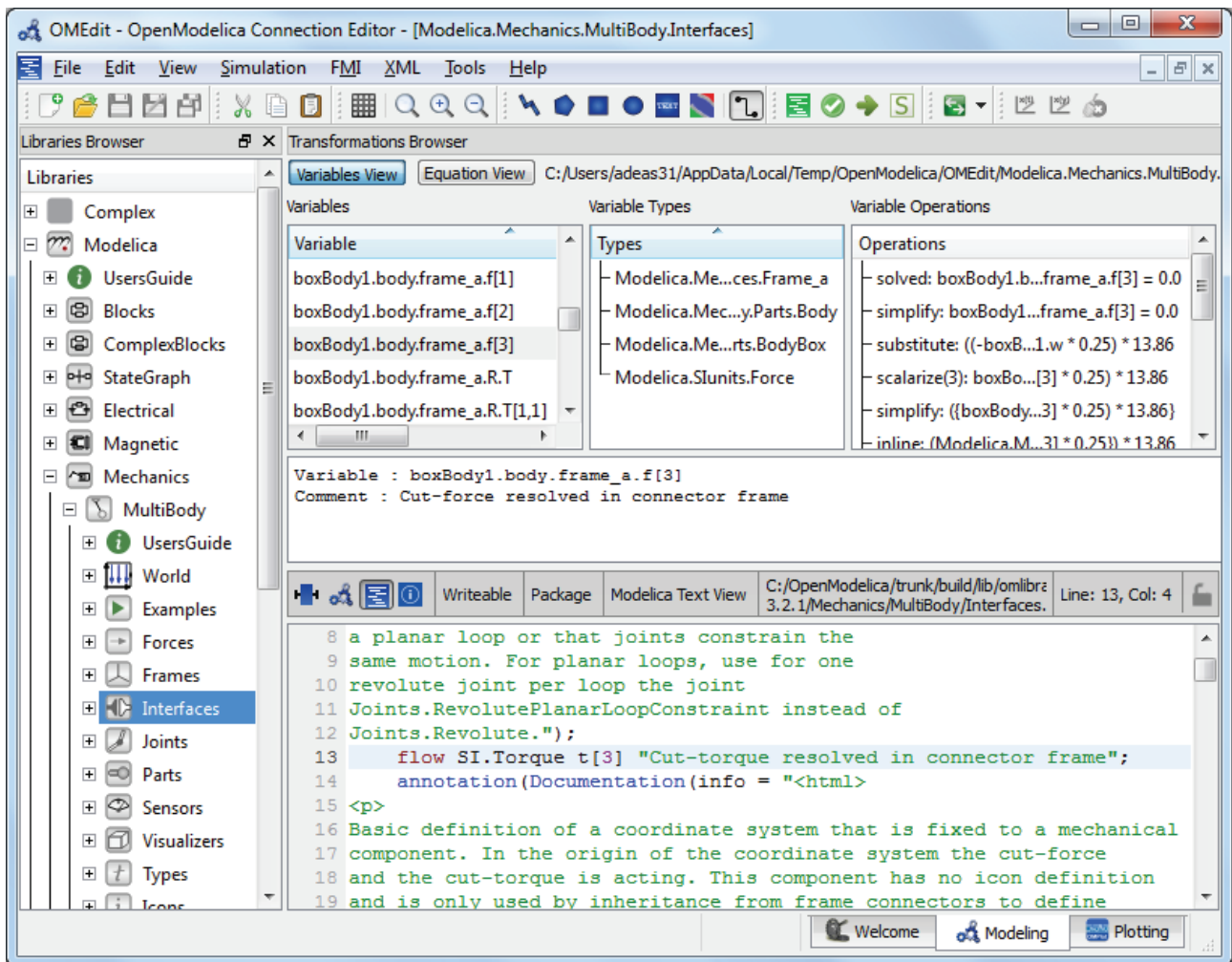
The transformations browser lists the variables along with their respective types hierarchy, operations performed, equations which defines the variable and equations which are using the variable. The types can be used to navigate to the specific class.

Double clicking on the equation updates the transformation browser and shows the list of operations and variables involved in the solution of the equation. See Figure 3.

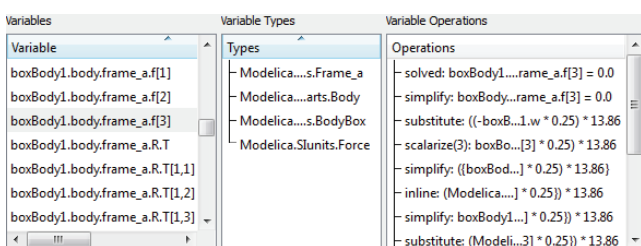
The transformation browser provides two views:

- Variables view
- Equation View

The data needed to build the structures shown in the GUI, i.e., the structural information about the equation systems, and the equation transformation traces, are loaded from an XML file which is generated by the OpenModelica compiler, see Section 4 for more details.

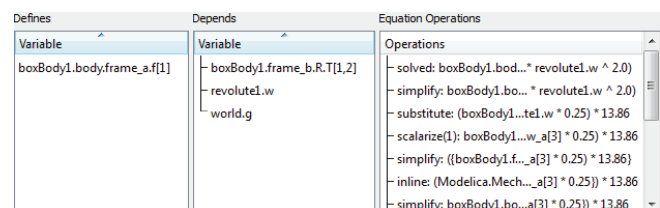


**Figure 1.** Transformations browser variables view with columns: Variables, Variable Types, Variable Operators.



**Figure 2.** Enlarged left part of variable info in transformations browser *variable view* with columns: Variables, Variable Types, Variable Operators.

When a numerical error is reported, clicking on the “Debug more” link at the end of the error report brings up the debugger showing the equation(s) involved in the error.



**Figure 3.** Enlarged part of transformation browser *equation view* with Defines variable, Depends on variable, Equation Operators operations like solved, simplify, substitute, etc.

## 4 Transformation Tracing

The underlying implementation of the transformation tracing mechanism is described in more detail in [7]. Some further improvements are present in this version.

The key idea introduced in [7] is to encode and store in a list all transformations that are performed by the Modelica compiler on the model equations, such as symbolic solution, alias elimination, symbolic differen-

tiation, etc. Because every operation is stored, it is possible to replay the operations and verify that the tool only performed sane operations during translation. This list of operations is then output to an XML-file (Figure 4) which is parsed by the debugger.

```

<simplify>
  <before>
    Nand.TP1.G.i + Nand.TN1.G.i + (-Nand.x2.i)
    = 0.0
  </before>
  <after>
    Nand.TP1.G.i + Nand.TN1.G.i - Nand.x2.i
    = 0.0
  </after>
</simplify>
<substitution>
  <before>
    Nand.TP1.G.i + Nand.TN1.G.i - Nand.x2.i
  </before>
  <!-- list of intermediate results -->
  <exp>0.0 + 0.0 - (-VIN2.i)</exp>
</substitution>
<simplify>
  <before>0.0 + 0.0 - (-VIN2.i) = 0.0</before>
  <after>VIN2.i = 0.0</after>
</simplify>
<solved>
  <lhs>VIN2.i</lhs>
  <rhs>0.0</rhs>
</solved>

```

**Figure 4.** List of equation transformations in the model `Modelica.Electrical.Analog.Examples.NandGate`.

The XML-file contains all the variables and equations used to solve the model, as well as variables that have already been solved for, alias relations, and so on. The equations are split into several groups, such as start-value equations, initial equations, regular equations, since the same variable may have different equations defined for it in different phases of the program.

These groups are related to how the compiler decided to numerically solve simulations. For example, the file includes the equations generated for the Jacobian, which is not used by all numerical ODE/DAE solvers.

Each equation knows the variables it solves for, as well as the variables it uses. This enables fast lookup of parents, children, and siblings in the BLT matrix. When reading the file, information is propagated to variables in a way such that each variable also knows the equation(s) where it is defined. This is again to ensure that the debugger can perform cheap lookup operations.

In the case of strongly connected components, an equation index will point to a set of equations (linear and nonlinear systems of equations in OpenModelica

are defined as a set of equations and variables to solve for). The generated code knows the index of an equation in the XML-file, so in case error or diagnostic messages are generated, a link to the equations and variables associated with this index can be provided to the debugger.

The message routines have been updated to take a list of equation indexes as an option, as well as output the messages as structured XML. This enables the debugger to read the messages and insert links to equations as appropriate.

This approach allows a user to debug simulations even if he/she did not run the simulation through the debugger, because it is possible to perform post-mortem debugging only based on the messages and diagnostics produced by the simulation executable.

There is no additional overhead during regular execution except reading and writing the additional information in the XML-file. This can be done by a thread running in the background and takes only a few seconds even for the large `EngineV6` model which both has many equations and many symbolic operations performed on each equation.

For error-messages there is an additional overhead of creating an error message that contains all the relevant information. This is a small one-time cost for error, which are hopefully infrequent. Consequently, the detailed error messages are output even if the user had not decided to debug the simulation before he started it since it will help him figure out why things went wrong.

## 5 Run-Time and Event Related Implementation

The run-time system performs the actual simulation of a Modelica model, in which the solution process is done by different solvers that cooperate in a master-slave hierarchical configuration, with the ultimate master being the end-user:

- ODE solver
- Functions computing the derivatives and algebraic variables
- Function computing the initial states and the values of parameters
- Function computing event points
- Linear equation solvers
- Nonlinear equation solvers

All of them may fail with different kinds of errors depending on the solver, generally because of numerical issues (e.g. singular Jacobian, no convergence, too tight

tolerance). However, at the bottom level they all share particular error types:

- Evaluation of expressions
  - Division by zero.
  - Functions called outside their domain (e.g.:  $\sqrt{-1}$ ,  $\log(-3)$ ,  $\text{asin}(2)$ ).
  - Evaluation of non-integer powers with negative argument
- Assertion violations for the model

In general some errors can be recovered automatically by the system (e.g. by re-trying with a shorter time step), whereas others abort the simulation and are reported to the user, which can then enter the debugging mode.

If an error cannot be recovered by the solver hierarchy, informative diagnostics are provided to the user. The diagnostic error message includes the corresponding equation block, the involved variables and their values. Furthermore the hierarchical context of the error is important to be able to classify it.

In the next step the user may be able to enter the debugging mode, where the simulation can be re-run to an accepted step just before the error occurs again. The last accepted step corresponds to the last point in time in the result file created in the first run. This point in time can be a breakpoint for debugging mode.

In the debug mode breakpoints are interpreted like zero-crossings, but without the time-consuming search process which the numerical solver does — the simulation just breaks if the condition becomes true.

Then the step that caused the failure is executed in a verbose mode, where informative diagnostic is provided for every equation that needs to be solved till the error occurs again. This allows the user to trace the solution process and if necessary, to engage by changing the model.

## 6 Example Models for Debugging

In this section some simple test cases are shown which demonstrate various possible error scenarios, and how a debugger can help their troubleshooting.

### 6.1 Chattering Models

In the model `ChatteringEvents1`, chattering takes place after  $t = 0.5$ , due to the discontinuity in the right hand side of the first equation. Chattering can be detected because lots of tightly spaced events are generated. The debugger allows to identify the equation from which the zero crossing function that generates the events originates.

```

model ChatteringEvents1
  Real x(start=1, fixed=true);
  Real y;
  Real z;
equation
  z = if x > 0 then -1 else 1;
  y = 2*z;
  der(x) = y;
end ChatteringEvents1;

```

Also in the model `ChatteringNoEvents1`, chattering takes place after  $t = 0.5$ , due to the discontinuity in the right hand side of the first equation. However, events are not generated in this case, because of the `noEvent` operator. If a variable-step-size integration algorithm with error control is used, the time step will be reduced to very small values once the discontinuity is hit, and this can be detected by monitoring the value of time at each time step.

The variable step size solver should be able to report which state variable(s) give the biggest contribution to the error estimate, thus causing the step size reduction. The corresponding derivative shows very high frequency oscillations between two values. The end user can then use the BLT navigation functionality of the debugger to investigate which variable/equation is introducing the discontinuity.

```

model ChatteringNoEvents1
  Real x(start=1, fixed=true);
  Real y;
  Real z;
equation
  z = noEvent(if x > 0 then -1 else 1);
  y = 2*z;
  der(x) = y;
end ChatteringNoEvents1;

```

Regarding `ChatteringFunction1`, after  $t = 0.5$ , chattering takes place due to the discontinuity in the right hand side of the first equation. The discontinuity is caused by a discontinuous function, which does not generate events.

The considerations regarding variable-step solvers, derivatives, and debugger BLT navigation are the same as for the previous example `ChatteringNoEvents1`.

```

model ChatteringFunction1
  Real x(start=1, fixed=true);
  Real y;
  Real z;

function f_sign
  input Real x;
  output Real y;
algorithm
  if x > 0 then
    y := 1;
  elseif x < 0 then
    y := -1;
  else
    y := 0;
  end if;

```



```

end f_sign;

equation
  z = Functions.f_sign(x);
  y = 2*z;
  der(x) = y;
end ChatteringFunction1;

```

## 6.2 Models with Different Numerical Failure Modes

The `NonlinearSolverFailureInitial` model describes a simple hydraulic system with a pump, followed by a valve, which fills a reservoir.

The initial value of the level of the reservoir is too high for the pump sizing, so the pressure `p2` is too high and consequently the nonlinear algebraic system of equations that determines `p1` and `w_pump` has no solution.

It is possible to find a solution to the system either by lowering the initial value of `y`, and thus the pressure `p2`, or by increasing the value of the parameter `dp0`, increasing the head the pump can provide.

The debugger can show the dependency of the nonlinear system of equations on the parameters `dp0`, `a1`, `a2`, `a3`, and `Kv` (also showing their values), as well as the dependency on `p2` (which has a too high value). Once one understands that `p2` is too high, it should be possible to continue the analysis, looking at the equation that determines `p2`, which in turn depends on the value of the state `y`, which is the root cause of the problem.

The nonlinear system that cannot be solved has five unknowns: `w_pump`, `dp_pump`, `dp_valve`, `sqrt_dp`, and `p1`, which can be easily reduced to one by using `dp_pump` as a tearing variable. The debugger can show the torn variables and the tearing variables, as well as the corresponding torn equations and implicit residual equations, and allows to track the values of all five variables during the iterations of the Newton algorithm.

```

model NonlinearSolverFailureInitial
  parameter SI.Pressure patm=101325
    "Atmospheric pressure";
  parameter Real Kv=1e-2 "Valve coefficient";
  parameter Real dp_small=1
    "Small dp for valve equation";
  parameter Real dp0=3e5 "Pump dp @ zero flow";
  parameter Real a1=1e6 "Pump coefficient";
  parameter Real a2=3e2 "Pump coefficient";
  parameter Real a3=3e2 "Pump coefficient";
  parameter SI.Temperature T0=20 + 273.15
    "Temperature of incoming fluid";
  parameter SI.Density rho=995
    "Density of fluid";
  parameter SI.Area A=0.01
    "Storage tank cross section";
  parameter SI.MassFlowRate w_extra=0
    "Extra mass flow rate into reservoir";
  constant SI.Acceleration g= 9.81
    "Acceleration of gravity";
  parameter SI.Temperature Tref=273.16
    "Reference temperature for specific

```

```

    enthalpy computation";
  parameter SI.SpecificHeatCapacity cp=4186
    "Cp of the fluid";
  SI.MassFlowRate w_pump
    "Mass flow rate from the pump";
  SI.Pressure p1 "Pump discharge pressure";
  SI.Pressure p2 "Storage tank inlet pressure";
  SI.Pressure dp_pump "Pump dp";
  SI.Pressure dp_valve "Valve dp";
  Real sqrt_dp "Regularized sqrt(dp)";
  SI.SpecificEnthalpy h0
    "Pump inlet specific enthalpy";
  SI.SpecificEnthalpy h1
    "Pump discharge specific enthalpy";
  SI.Power W "Pump power consumption";
  SI.Length y(start=40, fixed=true)
    "Reservoir level";
  Real eta(final unit="1") =
    (p1 - patm)*w_pump/rho/W "Pump efficiency";
  SI.Temperature T1
    "Pump discharge temperature";
  SI.Time tau=1
    "Time constant of temperature sensor";
equation
  dp_pump = p1 - patm "Pump dp";
  dp_valve = p1 - p2 "Valve dp";
  dp_pump = dp0 - a1*w_pump^2;
  w_pump = Kv*sqrt_dp;
  sqrt_dp = dp_valve/
    (dp_valve^2 + dp_small^2)^0.25;
  W = a2 + a3*w_pump;
  w_pump*(h1 - h0) = W;
  rho*A*der(y) = w_pump + w_extra;
  p2 = rho*g*y + patm;
  h0 = cp*(T0 - Tref);
  h1 = cp*(T1 - Tref);
end NonlinearSolverFailureInitial;

```

A simple modification of the previous model allows demonstration of the failure of the nonlinear solver in the causalization stage during simulation. The initial value of the level is reduced to 20, so that an initial solution can be found.

```

model NonlinearSolverSimulation
  extends NonlinearSolverFailureInitial(
    y(start=20), w_extra=0.2);
end NonlinearSolverSimulation;

```

In this case the reservoir is filled both by the pump and by an extra source. The mass flow rate of the pump `w_pump` is determined by a nonlinear system with five unknowns: `w_pump`, `dp_pump`, `dp_valve`, `sqrt_dp`, and `p1`, which basically computes the operating point of the pump as the intersection between the pump head curve and the load (valve + reservoir head) curve. Note that these curves have two intersections (also see `NonlinearSolverFailure3` later on). As the level increases, `w_pump` is reduced, and the two intersections get closer to each other, until at time  $t = 269$  they collide, making the system singular. As the level increases further due to the extra source, this system ceases to have any solution. This is a typical bifurcation pattern in nonlinear systems.

The debugger can show that the condition number of the Jacobian of the nonlinear system gets bigger and bigger as the critical time when the two operating

curves become tangent to each other, suggesting that this system becomes singular for some reason. Understanding the reason why this happens requires physical insight into the model.

The model can be fixed by adding some mass storage depending on the pressure  $p_1$ , in order to avoid the singularity in determining  $p_1$ , and also by using a more realistic cubic curve for the pump model, so that when the limit level is reached, the solution will jump to a big negative pump flow. Again, this requires physical insight into the validity range of the implemented model.

Another slight variation of the model allows demonstrating the case of finite escape time.

```

model FiniteEscapeTime
  extends NonlinearSolverFailureInitial(
    y(start=20));
  SI.Temperature Ts(start=T0);
equation
  tau*der(Ts) = T1 - Ts;
initial equation
  der(Ts) = 0;
end FiniteEscapeTime;

```

As the reservoir level increase, the flow rate  $w_{\text{pump}}$  goes to zero. When it does, the energy balance equation causes the specific enthalpy  $h_1$ , and thus the temperature  $T_1$ , to go to infinity.

The temperature  $T_1$  is the input of a first-order linear system, representing the temperature sensor dynamics. If a variable step-size solver with error control is used, it will try to compute the state trajectory, which also goes to infinity, so the solver eventually gets stuck at time  $t = 664$ .

If the ODE solver reports information on the state whose error estimate is causing the step size to be reduced, ( $T_s$ , in this case), then the debugger can point the end user to its derivative  $\text{der}(T_s)$ . It will be shown that it depends on  $T_1$ , whose values can be seen to grow indefinitely over time.  $T_1$  is shown to depend on  $h_1$ , which also goes to infinity. Finally,  $h_1$  depends on the energy balance equation, which depends on  $w_{\text{pump}}$ . At that point it will become apparent that as the flow rate  $w_{\text{pump}}$  goes to zero, the model becomes ill-posed. The solution in this case is to change the pump model, by adding to the energy balance some dynamic energy storage and/or some heat transfer to the ambient, in order to avoid the zero-flow singularity.

Finally, another small change to the original model presented in this section allows to demonstrate the debugging of models where the wrong initial solution is picked by the nonlinear solver.

```

model WrongInitialSolutionSelected
  extends NonlinearSolverFailureInitial(
    y(start=20),
    dp_pump(start=-1000));
end WrongInitialSolutionSelected;

```

The operating point of the pump is determined by a nonlinear system with five unknowns:  $w_{\text{pump}}$ ,  $dp_{\text{pump}}$ ,  $dp_{\text{valve}}$ ,  $\text{sqrt}_{\text{dp}}$ , and  $p_1$ . It is assumed here that  $dp_{\text{pump}}$  is selected as a tearing variable. At time  $t=0$ , this system has two solutions, one with positive  $w_{\text{pump}}$ , and the other one with negative  $w_{\text{pump}}$ . If the start value of the tearing variable  $dp_{\text{pump}}$  is chosen incorrectly, the solver will converge to the negative solution, then lock onto it for the rest of the simulation.

When the user sees the negative  $w_{\text{pump}}$  in the simulation (which is physically wrong), he/she should be able to analyze how this value was found at time  $t = 0$ . The debugger shows that  $w_{\text{pump}}$  is solved by that nonlinear system, and shows the values of the tearing variables and of the torn variables at each iteration step.

It will then become apparent that the start value of the tearing variable  $dp_{\text{pump}}$  leads to a negative value of the torn variable  $w_{\text{pump}}$ , leading to the solution of the problem, i.e., changing the start value of  $dp_{\text{pump}}$  to a value that allows convergence on the desired solution.

## 7 Background and Related Work

Modelica is a declarative language that makes writing equations easy while still producing efficient code. However, traditional debugging tools like GDB [12], Valgrind [19], or any of the other tools described in [18] assumes that the program being debugged is statement based. It also assumes that the user knows something about what the program is doing. This is fine if you are a Modelica compiler developer working on fixing some segmentation fault in your own code. A GDB-based approach exists for Modelica [4]; it works fine for debugging algorithms in functions.

But as a Modelica user you know very little about the internals of the run-time system. For example, there is speculative execution while simulating a model making debugging with GDB confusing.

There exists previous work on debugging in Modelica. Bonus [9] proposes a semi-automated dynamic (run-time) debugging of models where the user has to provide a correct diagnostic specification of the model which is used to generate assertions at runtime. Moreover, starting from an erroneous variable value the user explores the dependent equations (a slice of the program) and acts like an “oracle” to guide the debugger in finding the error.

Sjölund [7] is used as the main basis of the equation debugging part of this work. It was mainly focused on tracing operations in the compiler backend. It has been

extended with structured error messages from the simulation run-time system as well as an actual debugger.

Pop et al [3], [4] describe an integrated debugging approach based on a dependency graph. Edges in that dependency graph can be computed by the transformational tracing mechanism mentioned in Section 4.

## 8 Current Status

At the time of this writing, the implementation of the debugger framework in the OpenModelica environment is mostly complete but still missing some parts.

This debugger framework has three main parts: the tracing of symbolic operations in the backend of OpenModelica, reporting run-time errors in simulations, and the debugger implemented as an extension of the OMEdit graphical user interface.

The tracing of operations is complete, and the mapping of error positions in the low level generated code to the high-level model from where they originated. However, the reporting of run-time errors only works for a subset of problems at the moment.

The generation of the XML file with the transformation tracing, and its subsequent representation in the OMEdit GUI are fully implemented. Some types of numerical errors (e.g., chattering) can already be debugged as described in the paper.

However, the interface to the numerical solvers (both for the casualization and for the time integration steps) is still incomplete. Also the functionality of analyzing the results of simulation runs (which did not generate errors) at specific points in time is not implemented yet.

It is planned to have the implementation with the abovementioned additional functionality completed by fall 2014.

## 9 Conclusions and Future Work

We have presented a set of problems of simulating Modelica models that benefits from increased debugging tool support. We have also presented a design and implementation of the first (to our knowledge) documented debugging framework that can handle this set of problems.

The debugger is operational and has been tested on rather large models without noticeable run-time overhead. It is able to map error positions from low-level compiled simulation code to the corresponding source level equations in the Modelica model.

We believe that this kind of debugging support will significantly improve the ease-of-use regarding application modeling with Modelica compared to the current

situation typically needing a large amount of trial-and-error and a lot of expertise in the internal mechanisms of Modelica model compilers and simulation run-time systems. This can speed up the acceptance and use of Modelica in the engineering community.

Future work includes creating additional specialized debugging views including a view to display non-convergence of non-linear equation systems.

## 10 Acknowledgements

This work has been supported by the Swedish Strategic Research Foundation in the EDOp projects and Vinnova in the RTSIM and ITEA2 MODRIO projects, and by VR. The Open Source Modelica Consortium supports the OpenModelica work.

## References

- [1] Adrian Pop and Peter Fritzson (2005). A Portable Debugger for Algorithmic Modelica Code. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany.
- [2] Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani (2006). OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proc. of Modelica'2006*, Vienna, Austria.
- [3] Adrian Pop, David Akhvlediani, and Peter Fritzson (2007). Towards Run-time Debugging of Equation-based Object-oriented Languages. In *Proceedings of the 48th Scandinavian Conference on Simulation and Modeling (SIMS'2007)*, see <http://www.scan-sims.org>, <http://www.ep.liu.se>. Göteborg, Sweden.
- [4] Adrian Pop, Martin Sjölund, Adeel Asghar, Peter Fritzson, Francesco Casella. Static and Dynamic Debugging of Modelica Models. In *Proceedings of the 9th International Modelica Conference (Modelica'2012)*, Munich, Germany, Sept.3-5, 2012.
- [5] Martin Sjölund, Peter Fritzson, and Adrian Pop (2011a). Bootstrapping a Modelica Compiler aiming at Modelica 4. In *Proceedings of the 8th International Modelica Conference (Modelica'2011)*, Dresden, Germany.
- [6] Martin Sjölund and Peter Fritzson (2011b). Debugging Symbolic Transformations in Equation Systems. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, (EOLIT'2011), Zürich, Switzerland.
- [7] Martin Sjölund. *Tools for Understanding, Debugging, and Simulation Performance Improvement of Equation-based Models*. ISBN 978-91-7519-624-4, Linköping Studies in Science and Technology. Li-

- centiate Thesis No. 1592, ISSN 0280-7971, <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-90096>, May 27, 2013.
- [8] Peter Bunus and Peter Fritzson. Semi-Automatic Fault Localization and Behavior Verification for Physical System Simulation Models. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, Montreal, Canada, 2003.
- [9] Peter Bunus (2004). *Debugging Techniques for Equation-Based Languages*. PhD Thesis. Department of Computer and Information Science, Linköping University.
- [10] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press. 2004.
- [11] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman (2005). The OpenModelica Modeling, Simulation, and Software Development Environment. In *Simulation News Europe*, 44/45.
- [12] Richard Stallman, Roland Pesch, Stan Shebs, et al. (2011). Debugging with GDB. Free Software Foundation. [online] Available at: <<http://unix.lsa.umich.edu/HPC201/refs/gdb.pdf>> [Accessed 30 October 2011].
- [13] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. <http://www.openmodelica.org>
- [14] Modelica Association. *The Modelica Language Specification Version 3.2 revision 2*, July 30th 2013. <http://www.modelica.org>. Modelica Association. *Modelica Standard Library 3.2.1*. Aug. 2013. <http://www.modelica.org>.
- [15] Uri Ascher and Linda Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.
- [16] Willi Braun, Lennart Ochel, and Bernhard Bachmann. Symbolically derived Jacobians using automatic differentiation - enhancement of the OpenModelica compiler. In *Modelica'2011*.
- [17] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential algebraic equations using dummy derivatives. *Siam Journal on Scientific Computing*, 14:677--692, May 1993.
- [18] Andreas Zeller. *Why Programs Fail, Second Edition: A Guide to Systematic Debugging*. ISBN: 978-0123745156, 2009
- [19] Nicholas Nethercote and Julian Seward. Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '07. San Diego, California, USA, 2007, pp. 89-100. doi: 10.1145/1250734.1250746

# Making Modelica Applicable for Formal Methods

Matthew Klenk Daniel G. Bobrow Johan de Kleer Bill Janssen

Palo Alto Research Center  
3333 Coyote Hill Rd, Palo Alto, CA, 94304  
contact email: Matthew.Klenk@parc.com

## Abstract

Engineers need to perform many different types of analyses as they design systems. Modelica has become a leading language to support numerical simulation. As a consequence there is widespread understanding of Modelica and a large number of Modelica model libraries available. This paper addresses the task of using formal methods to derive system properties such as whether a design meets its requirements for all possible inputs. We report on our experience building a qualitative reasoner operating on Modelica models. In this paper, we highlight five Modelica modeling practices that impede the application of formal methods.

## 1 Introduction

Modelica [Fritzson, 2004] is a powerful language for specifying the behaviors of components represented by declarative constructs connected through power ports. Modelica provides designers with large libraries of standard models and compile time computation to create large models. These features attract designers interested in numeric simulation and researchers developing new analyses. In contrast, the languages qualitative reasoning [Weld and de Kleer, 1989] and other hybrid system verification methods (e.g., HybridSAL [Tiwari, 2012]) require equation-based models. Engineers use these formal methods to prove that systems will never reach critical states for all possible parameter values in a section of the design space. In previous work, we have discussed how qualitative reasoning can be used on Modelica models consisting only of a subset of the language [Klenk *et al.*, 2012].

In addition to this common core, Modelica allows designers to specify behavior using algorithms and provide advice for simulation engines. While designers desire this flexibility and control, these features make qualitative reasoning and other formal methods difficult to apply to Modelica models. In addition to simulation advice and explicit algorithms, we identify three other modeling practices that hinder the application of formal methods: unnecessary component model complexity, use of computational state, and incomplete models.

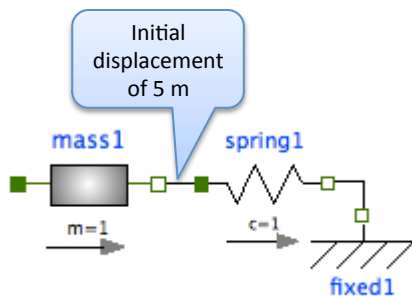
This paper is structured as follows. We begin with a brief overview of qualitative reasoning. Then, we discuss how the Modelica compiler may be used by formal methods. After which, we provide examples of each class of

hindrances along with suggestions for improving the applicability of Modelica models for formal methods. We close with a discussion of related work and some general reflections on modeling.

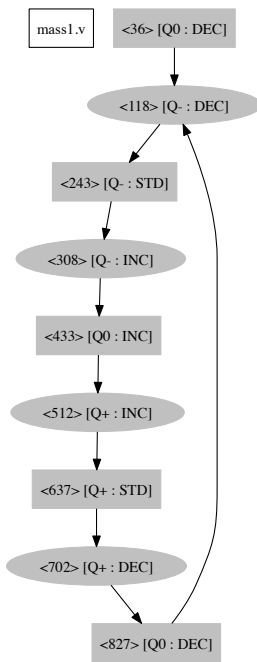
## 2 Qualitative reasoning and Design

Qualitative reasoning has its roots in automating reasoning about physical systems [Forbus, 1984][Kuipers, 1994][de Kleer and Williams, 1991]. Based on the intuition that engineers employ qualitative reasoning extensively throughout the design process, numerous researchers have sought to apply qualitative reasoning to design problems including functional reasoning [Everett, 1999][Wetzel and Forbus, 2009], diagnosis [Struss and Price, 2004], and automated FMEA generation [Snooke and Price, 2012]. An important subtask is *qualitative simulation* [Forbus, 1984][Kuipers, 1994], which provides an abstract description of the possible behaviors of a mathematical model. We illustrate qualitative simulation as well as some uses in the design context with a series of examples.

First, consider the spring block system in Figure 1 with the initial condition of a displacement of 5 meters compressing the spring. Given a set of numeric parameters and a simulation duration, Modelica produces a numeric simulation (i.e., a sequence of real values for each variable). On the other hand, qualitative simulation begins by creating a set of abstractions for each variable. The simplest set is three qualitative values (Q-, Q0, and Q+) corresponding to the sign of the real valued quantity. Each continuous variable can have as many higher-order derivatives as necessary, each of which specifies a direction of change ( $\downarrow$ (DEC),  $\rightarrow$ (STD),  $\uparrow$ (INC)) at that derivative order. Qualitative simulation determines all possible sequences of qualitative states a system can go through over time, called an *envisonment*. Changes in the qualitative state occur when a variable or its derivative reaches a *landmark* (e.g., the displacement of the block crosses zero, the velocity of the block crosses zero). Crossing a landmark occurs in an instant that has no duration (represented as a rectangle) and approaching or departing a landmark occurs over an interval of time (represented as an oval). The numeric simulation produces a sequence of numbers that must be interpreted by the designer to understand the behavior. On the other hand, the envisonment illustrates directly that this system is oscillatory because the graph is a loop (i.e., the system returns to the same qualitative state). Furthermore, while the numeric simulation results apply to specific values for the mass and compliance of



(a) Spring block system



(b) Envisionment with qualitative values for the velocity and its derivative are shown. Each state is assigned a unique id. Ovals represent qualitative states that exist over an interval of time and rectangles represent qualitative states that exist only for an instant.

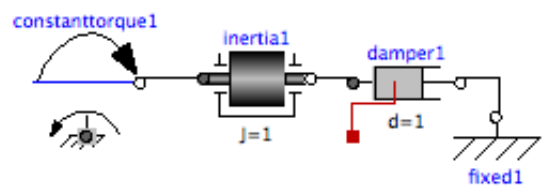
Figure 1: Qualitative simulation example

the spring, the envisionment illustrates that the behavior of the system in Figure 1 will be oscillatory for every set of parameters.

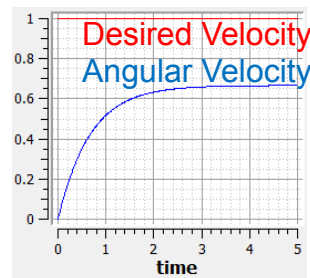
One reason for performing simulations is to determine if a system will meet some specified requirement. Figure 2 illustrates a simple rotational mechanical system with a specified requirement that the angular velocity must exceed 1 rad/s. This requirement can be encoded in Modelica using an enumerated type: Unknown, Success, Violated. The value of this variable begins as Unknown and changes to Success if the angular velocity of the flywheel exceeds 1 rad/s. As shown in the Modelica simulation (Figure 2), the system does not meet the requirement. Meanwhile, the envisionment (Figure 2) contains an interval in which the flywheel is accelerating followed by two branches: one where the requirement is met (shown as a green uparrow) and one where the inertia reaches

its asymptote (shown as a blue rectangle). This multi-trajectory simulation illustrates the range of behaviors that are possible given underspecified parameters (e.g., the moment of inertia, applied torque, and damping coefficient are known only to be positive). The Modelica simulation in Figure 2 corresponds to the the following trajectory of qualitative states: 48 → 122 → 313. Because this environment includes a trajectory in which the requirement’s value is Success, the designer knows that this topology may satisfy the requirement with different parameter values (e.g., increasing the torque or decreasing the damping factor).

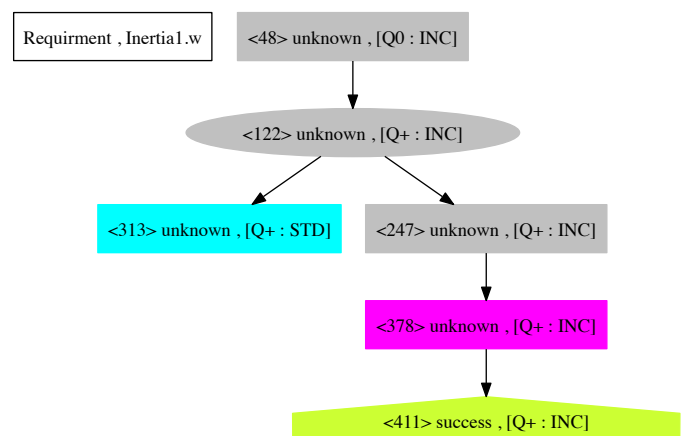
Figure 2: Qualitative Simulation and Requirements



(a) Modelica configuration with requirement that the flywheel reaches 1rad/s.



(b) Modelica simulation demonstrating that the current set of parameters does not meet the requirements within 5 seconds.



(c) The envisionment shows that this configuration could meet requirements with a different set of parameters. Cyan nodes represent terminal states, magenta nodes represent discrete events, and green uparrow nodes represent states that meet requirements.

Our intuition is that designers use this qualitative under-

standing of the design space to make decisions about components, parameters and configurations. Qualitative simulation is applicable early in the design process because it operates without completely specified parameters. Furthermore, qualitative simulation can be used to automate a number of the reasoning tasks designers perform during early design, answering such questions as:

- Could this configuration of components perform the desired function?
- What kinds of failures might this design have?
- How would this system behave when increasing a particular parameter?

Unfortunately, current qualitative simulation approaches are unable to make use of many existing Modelica models. In the rest of this paper, we discuss features of the Modelica language and practice that hinders the reuse of Modelica models by formal methods such as qualitative reasoning.

### 3 Modelica models for formal methods

Modelica's reuse and flexibility are central to its appeal among designers, engineers, and researchers. Unfortunately, these features create difficulties for applying formal methods. Some problematic features are: compiler interaction, artifacts for numeric simulation, unnecessary component model complexity, algorithms, sequential states, and incomplete models. For each issue, we will attempt to answer three questions:

- Why do designers use it?
- Why is this difficult for formal methods?
- What should be done to enable formal verification?

Before we discuss Modelica language issues, we discuss how Modelica compilers assist in our efforts to perform qualitative reasoning with Modelica models.

#### 3.1 Using the compiler to facilitate other analyses

Modelica tools include a compiler that takes as input a Modelica model and through a sequence of processes produces executable code to perform numeric simulation. Access to intermediate results during the compilation process facilitates other analyses. Here we highlight three aspects of the compilation process we have found useful.

##### Model construction language

Modelica has a powerful model construction language including iteration and conditional declarations. For example, the Damper model (shown in Figure 3) includes a conditional heat port, which allows the same damper model to be used in systems that consider thermal connections and those that do not. Also, compilers perform a number of optimizations on the model including index reduction and removal of redundant variables. These optimizations are applicable to qualitative reasoning as well. Therefore, our approach uses an XML representation of the hybrid-DAE produced by OpenModelica [Parrotto *et al.*, 2010]. Furthermore, we encourage the ongoing efforts to standardize an XML representation of the compiled model across Modelica tools.

##### System initialization

System initialization is a well-known difficult problem, and Modelica provides a number of language constructs to direct the solver to the initial state [Mattsson *et al.*, 2002]. These include the use of the `:start` and `:fixed` keywords, initial equations and initial algorithms. Qualitative simulation also requires knowing the initial values of the system variables. Therefore, we use OpenModelica to solve the initial equation system for a consistent set of initial values from which to begin our analysis.

##### Function inlining

Many Modelica functions are merely mathematical relationships between input and output variables. Consider the `from_kmh` function in the Modelica Standard Library shown in Figure 4. Function inlining is performed by many Modelica compilers to replace calls to these functions by their equivalent equations. The problems with analyzing arbitrary functions will be discussed in Section 3.4. Therefore, having the compiler perform these optimizations assists in translating Modelica models for use in formal methods.

Figure 4: Function that converts km/h to m/s

```
function from_kmh;
  input NonSIunits.Velocity_kmh kmh ;
  output Velocity ms "metre per second value";
algorithm
  ms := kmh/3.6;
end from_kmh;
```

#### 3.2 Artifacts of numeric simulation

Modelica is primarily a language for modeling and simulating mathematical models that evolve as a function of time. Consequently, there exist many constructs to assist with issues that arise in numeric simulation. While some are irrelevant for formal methods (e.g., `noEvent` and `smooth`), in this section, we discuss three that complicate formal methods analyses.

##### Equality involving continuous time variables

Modelica events occur at zero crossings. Therefore, it is not possible to have a condition testing equality of real valued variables,  $s1$  and  $s2$ . Instead, the `Modelica.math.IsEqual` function from the Modelica Standard Library which is computing using Equation 1.

$$result := abs(s1 - s2) <= eps; \quad (1)$$

While Equation 1 can be translated directly for formal methods, it both needlessly adds complexity to formal analysis and may yield unexpected results. Intuitively, `IsEqual` is testing whether  $s1$  and  $s2$  are equal. If the Equation 1 is directly encoded, the formal analysis will have to consider 7 cases: (1)  $s1 - s2$  is negative and more than  $eps$  from 0, (2)  $s1 - s2 = -eps$ , (3)  $s1 - s2$  lies between  $-eps$  and 0, (4)  $s1 - s2 = 0$ , (5)  $s1 - s2$  is positive and less than  $eps$ , (6)  $s1 - s2 = eps$ , (7)  $s1 - s2 > eps$ . In effect, it treats  $eps$  as an important parameter the system. As a consequence the number of states needed to be analyzed grows exponentially in the number of `IsEqual`'s translated in this way. Finally, the formal analysis could

Figure 3: Damper model from the Modelica Standard Library includes a conditional heat port connection that is set during model instantiation

```

model Damper "Linear 1D translational damper"
  extends Translational.Interfaces.PartialCompliantWithRelativeStates;
  parameter SI.TranslationalDampingConstant d(final min = 0, start = 0) "Damping constant";
  extends Modelica.Thermal.HeatTransfer.Interfaces.PartialElementaryConditionalHeatPortWithoutT;
equation
  f = d * v_rel;
  lossPower = f * v_rel;
end Damper;

```

produce incorrect results because it will accept as possible states in which  $s1$  is not equal to  $s2$  which is clearly against modeler's intent. Because formal methods allow for equality between continuous-time variables, the best solution is to simply translate `IsEqual(s1,s2)` to `s1 == s2`.

### Smoothing functions

Another piece of advice supplied by the model to the simulation engine concerns smoothing functions. For example, the function `Modelica.Fluid.Utilities.regStep` approximates a step function with a second order polynomial that is continuous and differentiable. Unless the transient behavior is the focus of the model, formal methods are more applicable to the idealized behaviors.

### 3.3 Unnecessary component model complexity

Modelers should be as concise and clear as possible when authoring models.

#### Optional model parameters

The inheritance features in Modelica make it easy to provide different variants of components that account for different behaviors. Therefore, in each model, every parameter should affect the behavior of the model. When this is not the case, the modeler has increased the complexity of the model unnecessarily. Consider the `w_small` parameter `PartialFriction` model. The default value of this parameter is  $1e10$  and the comment directs the engineer to set this to a small value if particular discontinuities are expected. This absurdly high value is to prevent it from affecting the simulation. Making these two separate models that inherit from the same model would facilitate formal methods by considering the `w_small` parameter only when it is necessary. Otherwise formal analysis will have to needlessly analyze the distinction between `w_small` and `w`.

#### Component modes

The evolutionary development of the Modelica language is apparent in the models of the Modelica Standard Library. For instance, many models use `Integer` variables to define a mode of operation for the model. However, these variables are typically unbounded, and often the default variable value of zero is not an applicable mode. Using enumerations would provide a definite set of modes of operation for these variables. However, even this is not sufficient. In some tool systems, such as `OpenModelica`, parameter variables of enumerated types are not required to be initialized to any particular value, and in that case

they default to the integer default value of zero, which is an invalid integer value for that enumerated type! In general, the semantics of operating modes, and more specifically enumerations (and parameters), seems to need more work in Modelica.

### 3.4 Imperative Code

Modelica algorithms can be executed at two times: flattening and simulation. All of the former algorithms pose no difficulty to formal analysis as they are executed before the DAE is created. Imperative code embedded in the DAE, typically in functions, to be executed at run time presents a fundamental challenge. Imperative code is important to model designers because certain numerical computations are easier to express as algorithms as opposed to equations. The Modelica language is Turing-complete, so proving properties of arbitrary Modelica programs is as hard as proving properties of any program. And proving properties of programs is a challenging intellectual field all to its own. Formal methods cannot be expected to analyze such algorithms. For common functions, we have created qualitative equivalents. For example, interpolation tables are essential to modeling complex physical systems, and, therefore, we have created an analogous concept for qualitative reasoning. There is independent interest in the Modelica community in elimination of imperatives when possible. For example, function inlining converts some imperatives to constraints automatically [Papadopoulos *et al.*, 2012].

We have applied our analysis technique to a wide variety of models. Too often we encounter needless imperatives. For example, consider:

```

Model Single_Clutch_Controller
  Output Real y;
  Input Integer u;
  parameter Integer num_gears = 5
  parameter
    Integer gear_nums[num_gears] = {-1,1,2,3,4}
  parameter Real engagement[num_gears]
    = {0.0, 1.0, 1.0, 0.0, 0.0};
algorithm
  y := 0.0;
  for i in 1:num_gears loop
    if u == gear_nums[i] then
      y := engagement[i];
    end if;
  end for;
end Single_Clutch_Controller;

model GBX_5_clutch_controller
  ...

```



```

Modelica.Blocks.Interfaces.RealOutput clutch_1;
Single_Clutch_Controller controller_c1
  (num_gears=num_gears,
   gear_nums=gear_nums,
   engagement=c1_eng);
...
equations
  connect(controller_c1.u, gear_selected);
  connect(controller_c1.y, clutch_1);
  ...
end GBX_5_clutch_controller;

```

Given a desired gear, the controller selects the clutch to activate. The algorithm block simply looks up the array index of the desired gear and reads off the clutch needed to engage. The clutch can be modeled as a table:

x	y
-1	0.0
1	1.0
2	1.0
3	0.0
4	0.0
X	0.0

Modelica has a table primitive (which begs the question why it wasn't used in this model). However, this particular table can be expressed as a Modelica equation:

```
y = if (x=1 or x=2) then 1.0 else 0.0;
```

To summarize, our approach to imperative code is:

- For widely used MSL functions such as interpolation we design them as primitives for the qualitative analysis. These function names appear in the XML DAE and thus can be treated as primitives.
- In limited cases, imperative code can automatically be translated to declarative code by function inlining.
- Key MSL models containing imperative code are being rewritten to be purely declarative (or use only known imperative primitives).
- User created functions and algorithms are currently not allowed. One unexplored possibility is the user must annotate the model specifying a piecewise linear approximation of the imperative code's behavior.

### 3.5 Sequential States

Many models include sources that iterate through a sequence of states (`Modelica.Blocks.Sources.SawTooth`) or components that exhibit delayed effects (`Modelica.Blocks.Logical.TriggeredTrapezoid`). The primary way this is handled in Modelica is by triggered Modelica events and setting a discrete-time real value representing the time at which the next state should change. Consider the variable  $T$  in the `TriggeredTrapezoid` model shown in Figure 5.

These models are difficult to analyze due to the non-local effects of the setting of the discrete-time variable. We propose to rewrite them without explicitly referencing time. This is done by using the events to set the rate and then the delayed effects occur when one of the continuous variables reaches a limit.

### 3.6 Incomplete models

Another complicating issue is that model authors frequently build models until their needs are met. While these models produce the simulation results the modeler expects, other analyses may have trouble using them due to untyped variables and operating regions.

#### Untyped variables

Modelica allows modelers to give types to variables, but modelers frequently use `Real` instead of more specific types. The Brake model in MSL defines `mue0` as type `Real` instead of `CoefficientOfFriction`. Automated fault modeling techniques (e.g., FAME [de Kleer *et al.*, 2013]) construct better fault models if variables are typed correctly.

### 4 Related work

The differences between the modeling languages used by engineers (e.g., Modelica, C++) and those used by model checking tools (e.g., finite state automata) hinder the deployment of formal methods in the design process. Researchers have taken both top-down and bottom-up approaches to overcoming this hurdle. Carloni *et al.* exemplify the top-down approach by arguing for a *semantic-aware interchange format* to make formal methods applicable across languages [Carloni *et al.*, 2006]. As an alternative to attempting to unify all hybrid systems languages, bottom-up approaches define automatic translations between subsets of pairs of languages. For example, Lundvall *et al.* translate a portion of Modelica models into hybrid automata for verification [Lundvall *et al.*, 2004]. Our approach follows the bottom-up tradition, and the contribution of this paper is a discussion of five Modelica modeling practices that hinder the automated analyses of Modelica models by formal methods.

### 5 Reflections on modeling

Modelica makes some fundamental semantic choices which are at odds with the formal methods, qualitative reasoning and cyber-physical systems communities. For example, Modelica allows one event to cause another — that is, two instants immediately following one another. Also, formal methods typically model behavior by modes, guards, and constraints. Modelica's guards and modes exist at the system level as conditions, but are not directly accessible from the models. So the common-sense engineering notion of mode has to be expressed by extra boolean variables and conditions. This can lead to very counterintuitive (to an engineer) models (e.g., the transistor models in MSL). For the purposes of our research, we have to accept Modelica's semantics. We do not know yet what problems this will cause.

Our experiences using Modelica models for other analysis purposes motivates some reflections on good modeling practices. In the course of this research, we have had to study and analyze a great many Modelica models. Some models in the MSL are diamonds, other models are disasters. We often wish that the Modelica community employed more standardized modeling practice. Modelica is such a general language that a modeler can write incredibly stupid models. Maybe that is because it is very hard to write clean, concise models. To summarize we suggest the following modeling principles:

Figure 5: Portion of the TriggeredTrapezoid model highlighting the use of discrete-time variables to initialize the timing of state transitions.

```

block TriggeredTrapezoid "Triggered trapezoid generator"
  extends Modelica.Blocks.Icons.PartialBooleanBlock;
  ...
protected
  ...
  discrete Modelica.SIunits.Time T
    "Predicted time of output reaching endValue";
equation
  y = if time < T then
    endValue - (T - time) * rate else
    endValue;
  when {initial(),u,not u} then
    ...
    T = if u and not rising > 0 or not u
      and not falling > 0
      or not abs(amplitude) > 0
      or initial() then
        time else
        time + (endValue - pre(y)) / rate;
  end when;
end TriggeredTrapezoid;

```

Figure 6: Portion of the TriggeredTrapezoidPARC model that explicitly states the conditions for state transitions.

```

block TriggeredTrapezoidPARC
  ...
protected
  Real rate;
equation
  when {initial(),u,not u,y>offset+amplitude,y<offset} then
    rate = ...
  end when;
  der(y) = rate;
end TriggeredTrapezoid;

```

1. Models should be easy to understand for both machines and people.
2. Models should be as declarative as possible, even when it's simpler to write imperative code. Such models are easier to understand by both people and machines.
3. Any "advice" to the compiler, especially dealing with small epsilon quantities in models should be done in standardized ways, such as `IsEquals`.
4. All variables should be declared by their physical type.
5. Models should have `assert` statements describing their range of applicability.

## 6 Acknowledgments

This work was partially sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program. Approved for Public Release, Distribution Unlimited. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official poli-

cies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

## References

- [Carloni *et al.*, 2006] Luca P Carloni, Roberto Passerone, and Alessandro Pinto. *Languages and tools for hybrid systems design*, volume 1. Now Pub, 2006.
- [de Kleer and Williams, 1991] J. de Kleer and B. C. Williams, editors. *Qualitative Reasoning about Physical Systems II*. Elsevier, Amsterdam, October 1991. *Artificial Intelligence* 51.
- [de Kleer *et al.*, 2013] Johan de Kleer, Bill Janssen, Daniel G. Bobrow, Tolga Kurtoglu, Bhaskar Saha, Nicholas R. Moore, and Saravan Sutharshana. Fault augmented modelica models. In *24th International Workshop on Principles of Diagnosis*, pages 71–78, Jerusalem, Israel, 2013.
- [Everett, 1999] John Otis Everett. Topological inference of teleology: Deriving function from structure via evidential reasoning 6. *Artificial Intelligence*, 113:149–202, 1999.

- [Forbus, 1984] K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1):85–168, 1984. Also in: Bobrow, D. (ed.) *Qualitative Reasoning about Physical Systems* (North-Holland, Amsterdam, 1984 / MIT Press, Cambridge, Mass., 1985).
- [Fritzson, 2004] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, Piscataway, NJ, 2004.
- [Klenk *et al.*, 2012] Matthew Klenk, Johan de Kleer, Daniel G. Bobrow, Sungwook Yoon, John Hanley, and Bill Janssen. Guiding and verifying early design using qualitative simulation. In *Proceedings of the ASME 2012 IDETC and CIE*, Chicago, IL, 2012.
- [Kuipers, 1994] Benjamin Kuipers. *Qualitative reasoning: modeling and simulation with incomplete knowledge*. MIT Press, Cambridge, MA, USA, 1994.
- [Lundvall *et al.*, 2004] Håkan Lundvall, Peter Bunus, and Peter Fritzson. Towards automatic generation of model checkable code from modelica. 2004.
- [Mattsson *et al.*, 2002] Sven Erik Mattsson, Hilding Elmqvist, Martin Otter, and Hans Olsson. Initialization of hybrid differential-algebraic equations in modelica 2.0. In *2nd Inter. Modelica Conference 2002*, pages 9–15, 2002.
- [Papadopoulos *et al.*, 2012] Alessandro V Papadopoulos, Martina Maggio, Francesco Casella, Johan Åkesson, and AB Modelon. Function inlining in modelica models. In *7th Vienna Conference on Mathematical Modelling*, 2012.
- [Parrotto *et al.*, 2010] Roberto Parrotto, Johan Åkesson, and Francesco Casella. An xml representation of dae systems obtained from continuous-time modelica models. In *EOOLT*, pages 91–98, 2010.
- [Snooke and Price, 2012] Neal Snooke and Chris Price. Automated {FMEA} based diagnostic symptom generation. *Advanced Engineering Informatics*, 26(4):870–888, 2012.
- [Struss and Price, 2004] Peter Struss and Chris Price. Model-based systems in the automotive industry. *AI Magazine*, 24(4):17–34, 2004.
- [Tiwari, 2012] Ashish Tiwari. Hybridsal relational abstracter. In *CAV*, pages 725–731, 2012.
- [Weld and de Kleer, 1989] D.S. Weld and J. de Kleer. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, 1989.
- [Wetzel and Forbus, 2009] Jon Wetzel and Ken Forbus. Automated critique of sketched mechanisms. *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena, CA*, 2009.



# Implementing stabilized co-simulation of strongly coupled systems using the Functional Mock-up Interface 2.0

Antoine Viel  
LMS Imagine  
7 Place des Minimes, 42300 Roanne, France  
antoine.viel@lmsintl.com

## Abstract

This paper addresses the main issue encountered with the co-simulation of coupled systems that exchange energy, i.e. the trade-off between computational performances and numerical stability. This property is first explained in details with the help of a simple generic test system for which a large oversampling with respect to the Nyquist frequency is required in order to keep a good level of accuracy. The linearly implicit stabilization method from [4] is then implemented and tested thanks to the directional directives computation capability of the FMI for Co-simulation 2.0 standard. Some minor extensions to the standard are proposed to efficiently implement the method. When applied to the test system, it is shown that large co-simulation steps can be taken, and hence significant computation time speed-ups are observed.

*Keywords:* Functional Mock-up Interface; co-simulation; linear system theory; stability

## 1 Introduction

Among the existing methods for coupling simulation models and software, co-simulation is used for performing transient simulations of coupled simulators<sup>1</sup>. The fundamental principle of co-simulation is to locally decouple in time simulators that are synchronized only through a limited set of coupling variables at scheduled time instants.

The most widespread numerical co-simulation scheme, is an explicit non-iterative Jacobi-type sequence of forward solving steps [7] done by each involved numerical solver (the stepping is described on

<sup>1</sup>A simulator being defined as the combination of a simulation model and mathematical libraries with a numerical solver, the latter being itself a combination of numerical integrators of ODE or DAE systems, error estimators, step size and order control heuristics, and discrete event schedulers.

figure 1 in the case of two simulators).

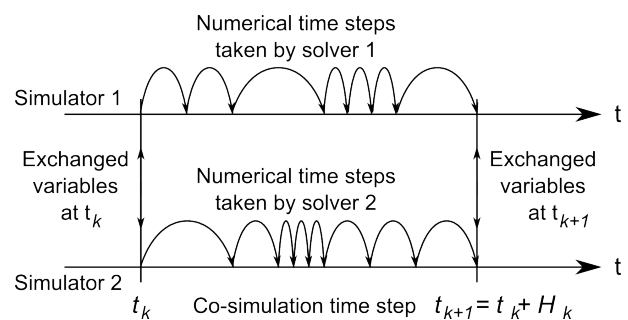


Figure 1: The explicit modular stepping scheme applied to the co-simulation of two coupled subsystems.  $H_k$  is the size of the current co-simulation or macro step.

As a consequence of this stepping scheme, each simulator is seen as a discrete dynamical system from the outside, and although each simulator is able to reach convergence if taken alone, the co-simulation process of energetically coupled systems is conditionally stable, even if the system is devoid of algebraic loops. This means that the convergence of the co-simulation process depends on the size of the co-simulation step, also called macro-step. An absolute stability limit does exist beyond which divergence is rapidly reached, and slightly below this limit undamped numerical oscillations still propagate between the subsystems. Divergence and poor accuracy are usually avoided by taking small enough macro-steps, which prevents the numerical solvers from taking large numerical micro-steps and thus leads to reduced computational efficiency with respect to continuously coupled systems that are integrated with a unique variable-step solver.

This property leads to a trade-off between computational efficiency and numerical stability which can be studied on the simple test case of a strongly coupled system introduced in the next section. In a latter sec-

tion, a numerical method is then implemented within the FMI for Co-simulation 2.0 framework, and tested with the same system, showing that this trade-off can be significantly enhanced.

## 2 A conditionally stable co-simulated system

### 2.1 Description of the test system

We consider a two degrees of freedom hydraulic system [1] obtained by connecting serially two elementary subsystems (see figure 2) made of:

1. a pipe modeled as a lumped-parameter nonlinear R-I element, with first-mode inertial effects and regular head losses
2. a volume modeled as a lumped-parameter nonlinear C-R element, with fluid-related compressibility effects, and singular head losses due to a leakage to the main circuit tank

The nonlinearities arise from the isothermal fluid properties that relate the density and compressibility to the system pressure, and from the laminar-turbulent friction models of the head losses. Some boundary conditions are introduced to model the surrounding environment: a constant pressure source on the left, and a transient flow rate source on the right.

### 2.2 Analysis of the continuously coupled system

The figure 3 shows the transient response of the continuously coupled system to the change of flow rate applied by the source on the right. The system parameters (pipe length and diameter, roughness, volume, head loss, ...) are chosen to be the same in the left and right subsystems. This choice is made to exemplify the nature of the coupling and its consequences on the dynamics of the coupled system.

Indeed, the dynamics of each subsystem can be studied by linearizing the system around some operating points, for example at the steady state following the first transient, for  $0.5 \leq t < 1$ . Instead of building the nonlinear state space and then evaluating the Jacobian matrix, a better understanding of the dynamics is obtained by analyzing the bond graph of the system.

Following the bond graph analysis of figure 4 and considering that the energy storage and dissipation elements are modeled using linear behaviour law, each elementary subsystem can be modeled by a first order

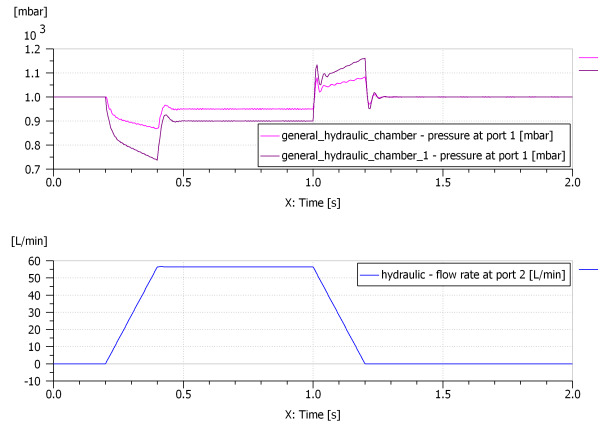


Figure 3: Transient pressure in the two hydraulic chambers (top) and source flow rate (bottom). Numerical simulation performed with the LSODA variable-step solver

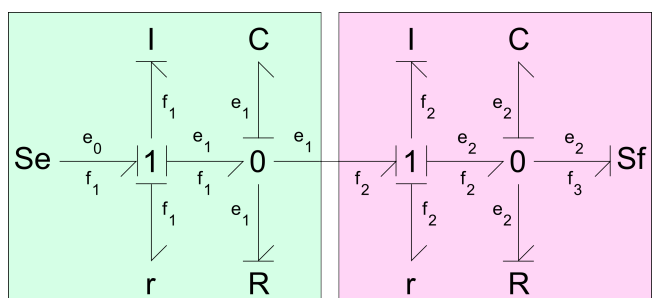


Figure 4: Bond graph of the system showing the partitioning in two subsystems. The coupling variables are  $e_1$  (output effort from first subsystem on the left) and  $f_2$  (output flow from second subsystem on the right).

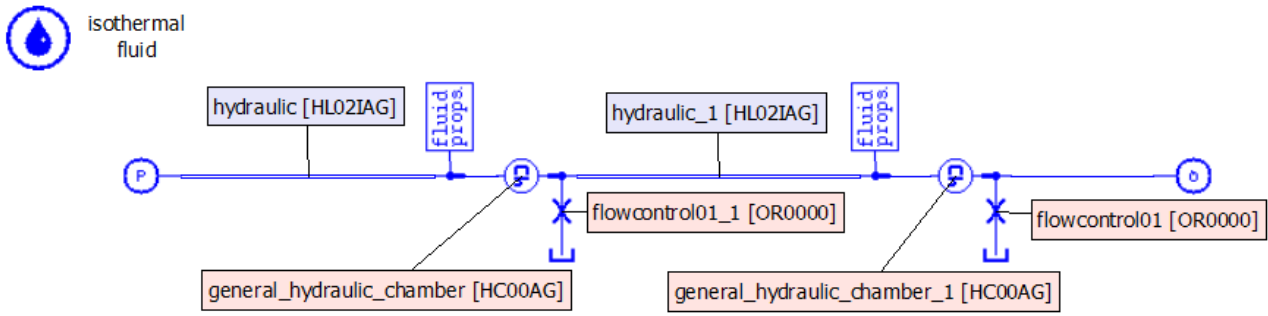


Figure 2: Sketch of an hydraulic two degrees of freedom system obtained by connecting two identical R-I-R-C subsystems built with LMS Imagine.Lab AMESim

linear system of ordinary differential equations in the state variables  $q_i$  and  $p_i$  associated with the C and I energy storage elements. For subsystem 1, the state-space equations are given by:

$$\begin{pmatrix} \dot{q}_1 \\ \dot{p}_1 \end{pmatrix} = \begin{pmatrix} -1/\tau & 1/I \\ -1/C & -2\zeta\omega_0 \end{pmatrix} \begin{pmatrix} q_1 \\ p_1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ f_2 \end{pmatrix} \quad (1)$$

where  $\omega_0 = 1/\sqrt{IC}$ ,  $\zeta = r/(2\omega_0 I)$  and  $\tau = 1/(RC)$  are the usual reduced parameters of a first degree of freedom linear system, and  $e_0, f_2$  are respectively the effort source of the left subsystem, and the flow source of the right subsystem. The output relation of subsystem 1 provides the effort  $e_1$  associated with the capacitive element:

$$e_1 = \begin{pmatrix} 1/C & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ p_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ f_2 \end{pmatrix} \quad (2)$$

For subsystem 2, it yields:

$$\begin{pmatrix} \dot{q}_2 \\ \dot{p}_2 \end{pmatrix} = \begin{pmatrix} -1/\tau & 1/I \\ -1/C & -2\zeta\omega_0 \end{pmatrix} \begin{pmatrix} q_2 \\ p_2 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} e_1 \\ f_3 \end{pmatrix} \quad (3)$$

where  $f_3$  is the flow source on the right of subsystem. The corresponding output relation gives the flow  $f_2$  that is also the input of subsystem 1:

$$f_2 = \begin{pmatrix} 0 & 1/I \end{pmatrix} \begin{pmatrix} q_2 \\ p_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} e_1 \\ f_3 \end{pmatrix} \quad (4)$$

For any subsystem, defined by equations (1) or (3), and provided that the damping parameters  $\zeta$  and  $1/\tau$  are small, the eigenvalues of the Jacobian matrix are

given by

$$\lambda_i = -\zeta\omega_0 \left(1 + \frac{1}{2\zeta\omega_0\tau}\right) \pm j\omega_0 \sqrt{1 - \zeta^2 \left(1 - \frac{1}{2\zeta\omega_0\tau}\right)^2} \quad \text{for } i = 1, 2 \quad (5)$$

whereas the eigenvalues of the whole system obtained by coupling the equations (1) and (3) through the output relations (2) and (4) are given by

$$\lambda_i = -\zeta\omega_0 \left(1 + \frac{1}{2\zeta\omega_0\tau}\right) \pm j\omega_0 \sqrt{\phi_i^2 - \zeta^2 \left(1 - \frac{1}{2\zeta\omega_0\tau}\right)^2} \quad \text{for } i = 1, 2 \quad (6)$$

where  $\phi_i^2 = \frac{3 \pm \sqrt{5}}{2}$  are the coupling coefficients. The non-unity ratio  $\frac{\phi_1}{\phi_2} = \sqrt{\frac{3+\sqrt{5}}{3-\sqrt{5}}} \neq 1$  expresses the fact that the two subsystems are strongly coupled and that part of the dynamics lie in the coupling itself.

### 2.3 Analysis of the co-simulated system

The nonlinear hydraulic system of figure 2 being partitioned in the two subsystems shown on 5, the resulting coupled system is co-simulated by assigning to each subsystem a slave simulator which embeds a numerical solver. With the explicit modular stepping shown on figure 1, the output variables of each subsystem  $e_1$  and  $f_2$  are sampled at the communication points  $t_k$ , and are taken as input variables which are held constant on the duration  $H_k$  of the macro-step.

Based on the eigenvalues (6), it should be possible to schedule the macro-steps  $(H_k)_{k=0, \dots, M-1}$ . A first approach consists in choosing a step size smaller than the Nyquist frequency, i.e. half the smallest time constant of the system, in order to ensure a proper sampling of

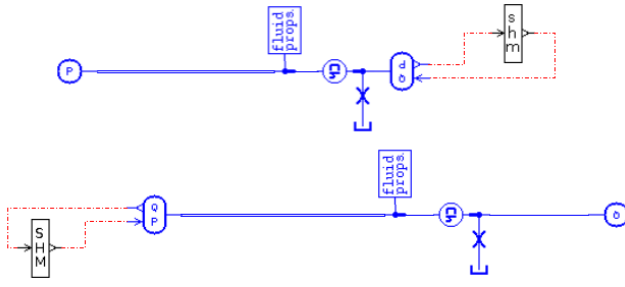


Figure 5: Sketch of the two nonlinear hydraulic subsystems being co-simulated within LMS Imagine.Lab AMESim. Blocks labeled *SHM* and *shm* are used for exchanging the coupling variables at the scheduled communication points. *SHM* stands for master simulator, whereas *shm* identifies the slave simulator.

the coupling variables:

$$H_k \leq \frac{\pi}{\max_{i=1,2} |\lambda_i|}$$

Unfortunately, this bound is too high regarding stability. This can be shown empirically by performing co-simulation with macro step sizes slightly smaller than the Nyquist bound (about one tenth of the above limit period, for small damping factors of less than 1 %). In a few macro-steps, instabilities propagate between the two subsystems that rapidly lead to divergence. To understand this phenomena and be able to correctly schedule the macro step size, it is necessary to fully analyze the stability of the system with coupling variables subjected to a zero-order sample and hold process, as shown on figure 6.

Stability analysis of this type of loop sampled system is carried by following the methodology described in [8]. This analysis, which focuses on the discretization of the coupling variables induced by the stepping, relies in other respects on the assumption that the subsystems can be exactly integrated by their respective numerical solvers<sup>2</sup>. Since there are only two subsystems connected through a unique loop, the analysis is done by considering the discrete-time transfer function associated with any of the two coupling variables which are obtained from the linearized<sup>3</sup> state-space

<sup>2</sup>or at least that these variable-step solvers are able to bound the truncation errors by any arbitrary tolerance

<sup>3</sup>around the same operating point reached in  $0.5 \leq t < 1$

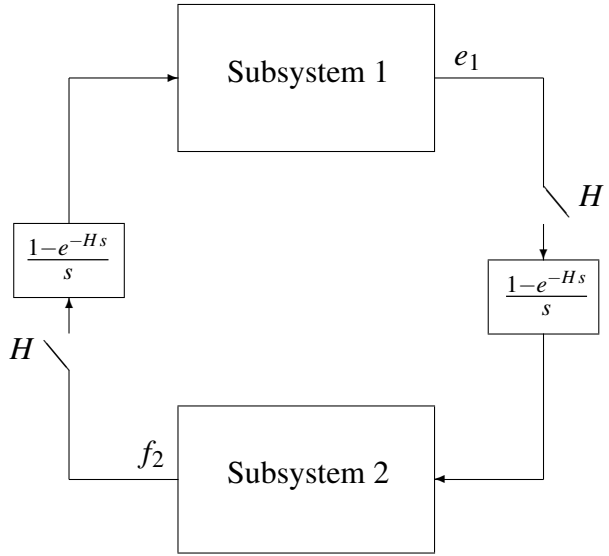


Figure 6: Bloc diagram of co-simulated system.  $H$  is the length of the current macro-step.

system (1)(2)

$$e_1(z) = -Z \left[ \frac{1-e^{-Hs}}{s} \frac{G_1(s)}{C} \right] f_2(z) + Z \left[ \frac{\omega_0^2 G_1(s)}{s + \alpha_1} e_0(s) \right] \quad (7)$$

or (3)(4)

$$f_2(z) = Z \left[ \frac{1-e^{-Hs}}{s} \frac{G_2(s)}{I} \right] e_1(z) + Z \left[ \frac{\omega_0^2 G_2(s)}{s + \alpha_2} f_3(s) \right] \quad (8)$$

with

$$G_i(s) = \frac{s + \alpha_i}{(s + \alpha_1)(s + \alpha_2) + \omega_0^2}$$

and

$$\alpha_1 = 2\zeta\omega_0, \quad \alpha_2 = 1/\tau$$

By combining the transfer functions (7)(8) and noticing that  $z = e^{Hs}$  by definition, the closed-loop transfer function is given by:

$$(1 + G^*(z)) e_1(z) = \omega_0^2 Z \left[ \frac{G_1(s)}{s + \alpha_1} e_0(s) \right] - \frac{\omega_0^2}{C} G_1^*(z) Z \left[ \frac{G_2(s)}{s + \alpha_2} f_3(s) \right] \quad (9)$$

in which

$$G_i^*(z) = (1 - z^{-1}) Z \left[ \frac{G_i(s)}{s} \right] \quad (10)$$



and

$$G^*(z) = \omega_0^2 G_1^*(z) G_2^*(z) \quad (11)$$

is the open loop discrete-time transfer function. Without giving here all the details about the calculation of (10) obtained using tables of Z-transforms, the Nyquist criterion [6] can be applied on the open loop transfer function (11) to evaluate the stability.

For given values of the reduced parameters, for example  $\zeta = 0.1\%$ , and  $\omega_0 \tau = 0.5$ , the Nyquist plot (figure 7) shows that the system is unstable for macro-step sizes such that  $H \frac{\omega_0 \phi_1}{\pi} \simeq 0.1$ .

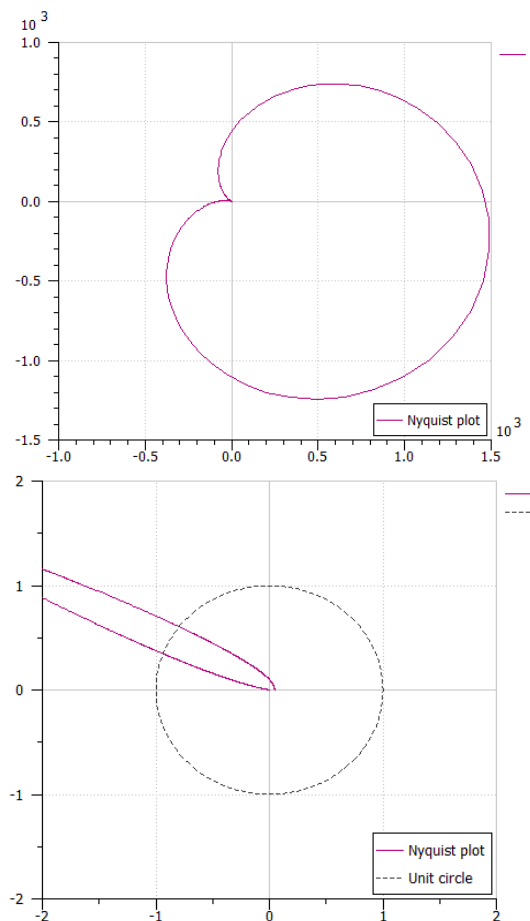


Figure 7: (top) Nyquist plot of the open loop transfer function for  $\zeta = 0.1\%$ ,  $\omega_0 \tau = 0.5$ , and  $H \simeq 0.1 \pi / \omega_0 \phi_1$ , which is a ten times oversampling of the Nyquist frequency. (bottom) Zoom on the unit circle showing the encirclement of the -1 point by the  $z = e^{j\omega H}$  contour for  $0 \leq \omega H \leq \pi$

With the same values of the damping parameters, it can be shown that the absolute stability limit is reached for  $H \frac{\omega_0 \phi_1}{\pi} \simeq 0.01$ , and a phase margin of at least  $2^\circ$  is obtained for a ratio less than 0.002, which means oversampling 500 times the Nyquist frequency. The effect of the damping coefficients on the phase margin is

analyzed on figure 8. This oversampling requirement

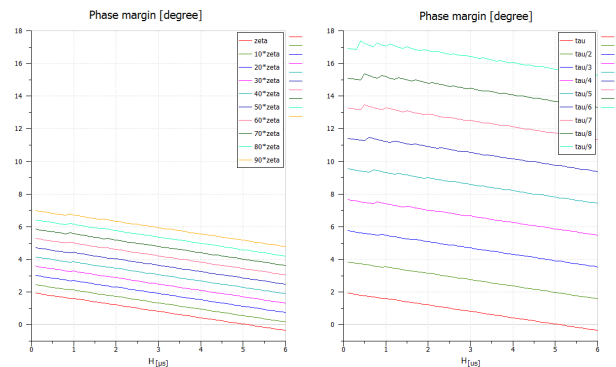


Figure 8: Plot of the phase margin versus macro-step size  $H$ , for different values of the the damping factor  $\zeta$  (left), and of the decay time constant  $\tau$  (right).

makes co-simulation unpractical for strongly coupled systems that are lightly damped. Indeed, comparison of the CPU time spent for performing the direct simulation of the continuously coupled system of figure 2, with the time needed for co-simulation with 500 times oversampling, exhibits a large slowdown, as shown on table 1. This is easily explained by looking at the number of micro-steps taken by the numerical solvers (last column of table 1). In the case of continuous simulation, the unique LSODA variable-step solver uses only 26000 steps, taking steps as large as  $76 \mu\text{s}$ , which is only a four times oversampling of the Nyquist limit period  $\frac{\pi}{\omega_0 \phi_1}$ . With co-simulation the local numerical solvers used for integrating each subsystem (DOPRI5 variable step solvers) cannot take large steps since the micro-step size is bounded by the macro-step size. It yields a large number ( $2 \times 10^6$ ) of dynamics function evaluation by the two numerical solvers, which is inefficient with respect to the frequencies of the coupled system.

### 3 Implementing the linearly implicit stabilization method

The rationale behind this method is to mitigate the stability-performance trade-off exemplified in the last section, by extending the phase margin. The linearly implicit stabilization method, first described by Arnold [4], makes use of the Jacobian matrices of the subsystems to build reduced linear models of the subsystems in state-space form that are exactly integrated locally in time using an unconditionally stable method. This allows to take relatively large macro-steps, which in

Type of simulation	CPU time [s]	Number of micro-steps	Maximum micro-step size [ $\mu$ s]
Direct simulation with LSODA variable-step solver	< 1	26 000	76
Co-simulation with $H = 1 \mu$ s ( $H \frac{\pi}{\omega_0 \phi_1} = 0.002$ )	70	2 000 000	1

Table 1: Comparison of continuous time simulation with co-simulation

turn do not restrict the size of the micro-steps taken by the embedded numerical solvers.

### 3.1 Description of the method

According to the FMI specification [5] the external representation of a system contained in a slave corresponds to a system of ordinary differential equations. Consequently the mathematical model of the whole coupled system is described by the following set of ODEs:

$$\dot{x}(t) = f(x(t), u(t)) \quad (12)$$

$$y(t) = g(x(t), u(t)) \quad (13)$$

$$u(t) = Ky(t) \quad (14)$$

where  $x(t) = (x_1(t), x_2(t), \dots, x_N(t))$  is the state vector obtained by gathering the state vectors of the  $N \geq 2$  subsystems being involved,  $u(t) = (u_1(t), u_2(t), \dots, u_N(t))$  and  $y = (y_1(t), y_2(t), \dots, y_N(t))$  are the input and output variables of all subsystems.

The third equation, which is not specified in the slave subsystems but in the co-simulation master, is required to close the above system. It defines how the output variables are connected to the input variables through a  $K$  matrix, which verify the following properties:

- it is a square matrix, since it can be assumed with no loss of generality that the output of a subsystem is connected to exactly one input of another subsystem
- the elements of  $K$  take their value in  $\{0, 1\}$
- there is exactly one 1 value per row and column of  $K$

The FMI specification [5] for Co-simulation allows a slave subsystem to expose its Jacobian matrices related to the equations (12-13):

$$\begin{aligned} A &= \nabla_x f(x, u) & B &= \nabla_u f(x, u) \\ C &= \nabla_x g(x, u) & D &= \nabla_u g(x, u) \end{aligned}$$

The linearly implicit stabilization method introduced by Arnold in [4] relies on the following three assumptions:

1. the product  $DK$  is assumed to be nilpotent [3], since the class of co-simulation methods considered here do not take into account algebraic loops on coupling variables
2. Inside a co-simulation macro-step, when  $t \in [t_k, t_k + H_k[$ , part of the system (12-13) can be approximated by a linear time invariant system obtained by linearizing the ODEs and the output relation around the point  $x = x(t_k)$ ,  $u = u(t_k)$
3. The linear approximate system is discretized using either the backward Euler method or the trapezoidal rule [2]

Assumption 2 leads to consider the following linear system:

$$\dot{\xi}(t) = \dot{x}(t_k) + A \xi(t) + B(w(t) - u(t_k)) \quad (15)$$

$$\eta(t) = y(t_k) + C \xi(t) + D(w(t) - u(t_k)) \quad (16)$$

where  $A, B, C, D$  are obtained at  $t = t_k$  and  $\xi$ ,  $\eta$  and  $w$  are the counterpart of  $x$ ,  $y$  and  $u$  in the linear system. With this choice of variable, the corresponding initial condition is given by  $\xi(t_k) = 0$ .

The coupling equation (14) is thus rewritten to couple the dynamic system of each subsystem with the approximate linear system, inside a macro-step:

$$u(t) = K \eta(t) \quad (17)$$

$$w(t) = Ky(t) \quad (18)$$

On the duration of a macro-step, the differential algebraic system made of equations (12), (15), (13), (17), (16), (18) holds. As there is no algebraic loop (assumption 1), this DAE can be reduced to a coupled set of ODEs by taking the derivate the last four equations.

Following assumption 3, the ODE (15) is first discretized using one of the two proposed methods:

$$(I - r(t - t_k)A) \xi(t) = (t - t_k) [\dot{x}(t_k) + rB(w(t) - u(t_k))]$$

where  $r$  depends on the discretization method and is either 1 for backward Euler or 0.5 for the trapezoidal rule. This equation provides an estimate for the state vector derivative:

$$\dot{\xi}(t) \simeq \mathcal{A}(t) [\dot{x}(t_k) + rB(w(t) - u(t_k))]$$

as well as for the output relation (16):

$$\begin{aligned} \dot{\eta}(t) &= C \dot{\xi}(t) + D \dot{w}(t) \\ &= C \mathcal{A}(t) [\dot{x}(t_k) + rB(w(t) - u(t_k))] + D \dot{w}(t) \end{aligned} \quad (19)$$

with

$$\mathcal{A}(t) = (I - r(t - t_k)A)^{-1}$$

In order to explicitly take into account the lack of algebraic loop, the equation (18) is approximated using the assumption 2. This means that the inputs  $w$  are obtained from a linear approximation of the outputs  $y$  of the subsystems around the point  $t = t_k$ :

$$w(t) - u(t_k) = K[C(x(t) - x(t_k)) + DK(\eta(t) - y(t_k))]$$

$$\dot{w}(t) = K[C\dot{x}(t) + DK\dot{\eta}(t)]$$

After substituting these two relations into (19) and noticing the nilpotency of  $DK$  due to assumption 1, an explicit differential equation for the outputs  $\eta$  is obtained:

$$\begin{aligned} \dot{\eta}(t) &= C \mathcal{A}(t) \dot{x}(t_k) + DKC f(x(t), K\eta(t)) \\ &+ rC \mathcal{A}(t) BK [C(x(t) - x(t_k)) + DK(\eta(t) - y(t_k))] \end{aligned} \quad (20)$$

The coupling condition (17) applied to (12) finally rewrites as:

$$\dot{x}(t) = f(x(t), K\eta(t)) \quad (21)$$

These two ODE (20) and (21) along with the initial condition  $\eta(t_k) = y(t_k)$  explicitly define the dynamics of the system on the duration of a macro-step  $[t_k, t_{k+1} = t_k + H_k]$ . At the end of the step the outputs are evaluated and propagated among the subsystems using (13):

$$y(t_{k+1}) = g(x(t_{k+1}), K\eta(t_{k+1}))$$

### 3.2 Computational flow

The following notation is introduced to describe the submatrices obtained by restricting to the variables involved in the slave simulator numbered  $s \in \{1, \dots, N\}$ :

- $K_{s,s}$  is the square submatrix of  $K$  obtained by taking the columns corresponding to the output variables of slave  $s$ , and the rows corresponding to the input variables of the other slave simulators that are connected to the outputs of slave  $s$ ;
- $K_{s,\bar{s}}$  is the square submatrix of  $K$  obtained by taking the rows corresponding to the input variables of slave  $s$ , and the columns corresponding to the output variables of the other slave simulators that are connected to the inputs of slave  $s$ .

The computational flow is a two steps process, the first step taking place at the communication point of co-simulation, the second step being the continuous time solving of the coupled DOE system (20)(21) during one co-simulation macro-step.

**At  $t = t_k$ :** The slave subsystem  $s \in \{1, \dots, N\}$  computes:

- the derivative of its state vector

$$\dot{x}_s(t_k) = f_s(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k))$$

- its outputs

$$y_s(t_k) = g_s(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k))$$

- the Jacobian matrices

$$\begin{aligned} A_{s,s} &= \nabla_x f_s(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k)) \\ B_{s,s} &= \nabla_u f(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k)) \\ C_{s,s} &= \nabla_x g_s(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k)) \\ D_{s,s} &= \nabla_u g_s(x_s(t_k), K_{s,\bar{s}} \eta_{\bar{s}}(t_k)) \end{aligned} \quad (22)$$

- it also provides the updated value  $x_s(t_k)$  of its state vector. If  $k = 0$  this is the global initial condition of (12)
- it receives its inputs and set up the stepwise local initial condition for its extended state:

$$K_{s,\bar{s}} \eta_{\bar{s}}(t_k^+) = K_{s,\bar{s}} y_{\bar{s}}(t_k) \quad (23)$$

**For**  $t_k < t \leq t_{k+1}$ : The slave subsystem  $s \in \{1, \dots, N\}$  solves a subset of the coupled DOE system (20)(21):

$$\dot{x}_s(t) = f_s(x_s(t), K_{s,\bar{s}} \eta_{\bar{s}}(t)) \quad (24)$$

$$K_{s,\bar{s}} \dot{\eta}_{\bar{s}}(t) = \bar{f}_{\bar{s}}(x_s(t), K_{s,\bar{s}} \eta_{\bar{s}}(t), t) \quad (25)$$

where  $\bar{f}_{\bar{s}}$  is evaluated by the master:

$$\begin{aligned} \bar{f}_{\bar{s}}(x_s, K_{s,\bar{s}} \eta_{\bar{s}}, t) = & K_{s,\bar{s}} C_{\bar{s},\cdot} \mathcal{A}(t) \dot{x}(t_k) \\ & + K_{s,\bar{s}} D_{\bar{s},\bar{s}} K_{\bar{s},s} C_{s,s} f_s(x_s, K_{s,\bar{s}} \eta_{\bar{s}}) \\ & + r K_{s,\bar{s}} C_{\bar{s},\cdot} \mathcal{A}(t) B_{\cdot,\bar{s}} K_{\bar{s},s} [C_{s,s} (x_s - x_s(t_k)) \\ & + D_{s,s} K_{s,\bar{s}} (\eta_{\bar{s}} - y_{\bar{s}}(t_k))] \quad (26) \end{aligned}$$

Practically, the derivative of the state vector  $\dot{x}_s$  is provided by the slave to the master, which uses it for evaluating the second term of equation (26) instead of evaluating the function  $f_s$  that appears in the right-hand side of equation (24).

### 3.3 Implementation within the FMI framework

The guiding principle behind the organization of computation is a strict devolution of responsibility between the slaves and the master in the co-simulation process. Each slave FMU is responsible locally for the system being solved and does not have any information about the coupling and the surrounding environment. It is up to the master algorithm to gather and assemble this information from the different slaves and to provide to the slaves ways for evaluating the additional ODE that represent the linearized part of the coupled system. This information is provided partly in the description<sup>4</sup> of the model structure of each slave, and partly at run-time through appropriate functions that evaluate the directional derivatives of the system enclosed in the FMU. With this structure-related information, as well as the variables exchanged at communication like the outputs of the models, state variables and their derivatives, a cooperation between the master simulator and slave simulators can be set up that keep the organization clean from the point of view of computational responsibilities.

In addition, the implementation of the stabilized co-simulation has to be compatible with the classical modular stepping specified by the FMI for Co-simulation [5]. This means that the stabilization method operates only if both the slaves FMU and the

co-simulation master cooperate. If it is not supported by any part of the coupled systems, the classical modular stepping with stepwise extrapolation has to be applied.

On the side of the slave FMU, it is mandatory to first enable the computation of the directional derivatives of all state variables and connected outputs, with respect to all state variables and connected inputs. This is done by declaring the flag *providesDirectionalDerivative* and by implementing the *fmiGetDirectionalDerivatives* function of the FMI specification. Moreover, the stabilization method being a model-based extrapolation, it is not compatible with the optional history-based extrapolation schemes that are allowed by the FMI specification. So the current *canInterpolateInputs* attribute has to be extended to specify which type of extrapolation is actually supported.

The way the slave handles its input variables has to be modified: the input variables that appear in the dynamics equation (24) are now considered as additional state variables, according to equation (25), and the input variables now act as initial conditions (equation (23)) for these state variables. The state vectors to be solved by the numerical integrator embedded in the slave is thus  $(x_s, u_s)$ , where  $u_s$  is actually  $K_{s,\bar{s}} \eta_{\bar{s}}$ , the actual mapping between the outputs  $\eta_{\bar{s}}$  of the linear system and the inputs of the slave being done by the master, since the structural information about connection (i.e. the elements of the  $K$  matrix) is only known by the master simulator.

If the master does not support stabilization, the right-hand side of equation (25) reduces to zero and so the actual inputs of the slave do not vary: a zero-order hold extrapolation is thus performed as defined in the FMI specification when the *canInterpolateInputs* flag is not set. In that case, the inputs of each slave is directly given by the initial condition (23), in which  $y_{\bar{s}}$  are the outputs of the slaves that are fed to the inputs of the slave number  $s$ , through the *fmiGetReal* and *fmiSetReal* functions called by the master simulator at each communication point  $(t_k)_{k=0,\dots,M-1}$ .

On the side of the master simulator, more tasks are required to implement the linearly implicit stabilization method. The master has to set up a callback function that is used to evaluate the dynamics associated with the inputs of the slaves. Before co-simulation, the model structure information of the slaves is used to prepare the matrices and the computations that appear in (26). Then, during co-simulation, these elements are updated at each communication step, through calls to the *fmiGetContinuousStates*, *fmiGetDerivatives* and

<sup>4</sup>stored in the corresponding XML file distributed with the FMU

*fmiGetDirectionalDerivatives* functions. Notice that the first two functions are originally defined only in the *Model Exchange* part of specification, so the Co-simulation specification has to be extended in the future. The same is true about the availability of the callback function associated with relation (26). This function is aimed at being called by the numerical solvers of the slaves, when performing the numerical integration of (24)(25). A proposal for extending the FMI for Co-simulation is to define a function called for example *fmiGetStabilizedInputDerivatives* and having the following arguments:

- the current time of the slave numerical solver at which the right hand side of (26) is desired
- the current value of the additional state variables, i.e. the stabilized inputs  $u_s = K_{s,\bar{s}} \eta_{\bar{s}}$
- the current values of the state variables and state derivatives  $(x_s, f_s(x_s, u_s))$  of the slave
- practically, a reference to the master environment, declared as *componentEnvironment* during instantiation, may be needed to help the master simulator identify the calling slave.

This function should return the vector of derivatives of the stabilized inputs, according to relation (26). It has to be declared by the master environment in the *fmi-CallbackFunctions* argument of the slave instantiation function.

The master algorithm is roughly sketched in Algorithm 1, in which the additional tasks required for stabilization are emphasized.

### 3.4 Test results

The hydraulic test case studied in section 2.1 and composed of two elementary hydraulic systems connected in serie is tested under various conditions of co-simulation:

- Reference implementation : transient simulation of the continuously coupled system, using the LSODA variable-step solver [2]
- Co-simulation with native interfaces in the LMS Imagine.Lab AMESim simulation environment, or through a specially crafted prototype of a master simulator supporting the FMI 2.0 RC1 [5] specification with the proposed extensions described in section 3.3

**Require:**  $N$  slave FMU  $s \in \{1, \dots, N\}$  and a sequence of  $M$  macro-steps  $\{H_0, \dots, H_{M-1}\}$ .  
 read the model structure description of slaves and create the connection matrix  $K$   
 instantiate each slave  $s$  and provide to it a callback function  $f_{\bar{s}}$   
 provide the initial conditions for all slave variables  
 initialize the slaves  
 initialize time  $t = t_0$   
**for**  $k = 0$  **to**  $M - 1$  **do**  
   get the outputs of slaves in  $y$   
   get the state variables and derivatives  $x_s, \dot{x}_s$  of slaves  
   get the Jacobian matrices  $A_{s,s}, B_{s,s}, C_{s,s}, D_{s,s}$  of slaves  
   set the inputs of slaves as  $u = Ky$   
   perform one macro-step for the slaves from  $t$  up to  $t + H_k$   
   update time  $t \leftarrow t + H_k$   
**end for**

**Algorithm 1:** Master simulator algorithm. Lines in blue correspond to the additional tasks required by the stabilization method.

- Comparison of stabilized co-simulation with explicit modular stepping
- Comparison of a mixed C/Python implementation for the co-simulation master, or «direct» C implementation, both with DOPRI5 variable-step solver for the slave simulators

For comparison purpose, all tests are performed with the same values of the reduced damping coefficients as in section 2.3, namely  $\zeta = 0.1\%$  and  $\omega_0 \tau = 0.5$ . The limit period corresponding to the Nyquist frequency is  $\pi/\omega_0 \phi_1 \simeq 425 \mu\text{s}$ , the absolute stability limit is reached for  $H \simeq 5 \mu\text{s}$ , and a  $2^\circ$  phase margin is obtained for  $H = 1 \mu\text{s}$ .

The test results are summarized in table 2. The accuracy of co-simulation is measured by taking the root mean square error on the coupling variables  $e_1$  and  $f_2$  with respect to the reference implementation and the mixed tolerance for all the variable-step solvers is set to  $10^{-5}$ . In all co-simulation tests, the CPU time is measured by setting the print interval (i.e. the sampling of result variables) to  $50 \mu\text{s}$ , corresponding to the largest macro-step size used across the tests, in order to have comparable processing times regarding result storage and disk access.

The stabilization method allows to choose large macro-step sizes  $H$ , up to the size of the numerical

Tested implementation	Macro-step size $H$ [ $\mu\text{s}$ ]	RMS error	CPU time [s]
#1. Continously coupled system with variable step solver in AMESim		<i>reference</i>	1.0
#2. Explicit co-simulation, native AMESim interface	1	104	70.0
#3. Stabilized co-simulation, native AMESim interface	50	207	8.5
#4. FMI 2.0 explicit co-simulation, mixed Python/C prototype	1	123	40.0
#5. FMI 2.0 stabilized co-simulation, mixed Python/C prototype	50	237	36.0
#6. FMI 2.0 explicit co-simulation, direct C prototype	1	123	12.0
#7. FMI 2.0 stabilized co-simulation, backward Euler, direct C prototype	50	215	1.5
#8. FMI 2.0 stabilized co-simulation, trapezoidal rule, direct C prototype	50	152	1.5

Table 2: Summary of tests performed with different implementations, macro-step sizes  $H$  and discretization methods.

micro-steps taken by the variable-step solver in the reference case. Indeed, the absolute stability limit of co-simulation seems to be reached for macro-step sizes of about  $76 \mu\text{s}$ , which is the maximum numerical step size reported in table 1. Consequently, a maximum value of  $50 \mu\text{s}$  is used for comparison across the tests #3, #5, #7 and #8.

Clearly, the performances obtained for the maximum macro step size depend on the implementation. Additional operations are needed by the stabilization method like the computation of Jacobian matrices and integration of the additional state equations related to the inputs on the side of the slaves, and the evaluation of the approximate dynamics (26) on the side of the master. This overhead, if not efficiently implemented, may cancel out the gain over the number of micro-steps. This is the case with the mixed Python/C implementation #5 of the master simulator, which exhibits poor performances with respect to #3, due to the too many context switches between the two environments. On the contrary, the two other C-based implementations #3 and #7 (or #8), show a large speed-up factor of about 8 with respect to the corresponding explicit co-simulations #2 and #6.

Regarding the accuracy, the tests #3, #5 and #7, which are based on a backward Euler method, yield nearly the same level of accuracy, about twice the error obtained with explicit co-simulation at  $H = 1 \mu\text{s}$ . The increase of the RMS error with the macro-step size is depicted on figure 9 for tests #7 and #8. Clearly, the use of a second-order method like the trapezoidal rule provides more accurate results, for the same computational load.

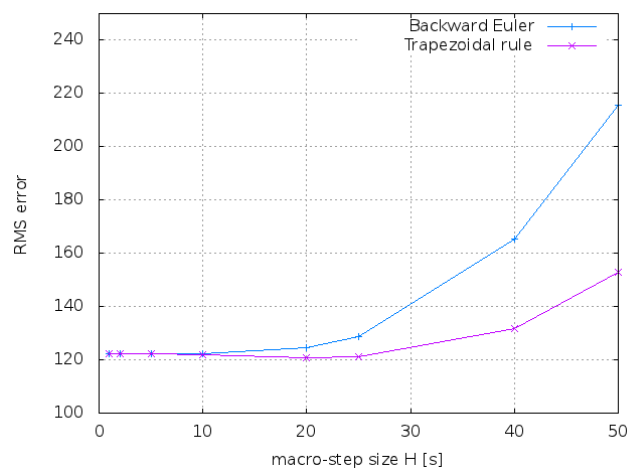


Figure 9: RMS errors obtained with respect to continuously coupled simulation versus the macro step size.

## 4 Conclusions

Although co-simulation is generally considered a robust method of simulator coupling, this paper presented the main issue that remains with the co-simulation of strongly coupled system, namely the trade-off between stability (or accuracy) and the computational performances. With the help of a simple yet representative example of this class of system, it showed how the stability issue may affect the computational load, since an oversampling factor as large as 500 is observed with respect to the highest dynamics of the system. The implementation of the linearly implicit stabilization method within the framework of the FMI for Co-simulation 2.0 standard then showed that significant speed-up can be regained at the price of a moderate loss of accuracy, provided that an efficient implementation is available as well as some minor extensions to the FMI for Co-simulation specification. With the advent of the FMI 2.0 specification, which one of its major enhancements is an interface for the directional derivative matrices, it seems that the efficient co-simulation of strongly coupled systems becomes feasible. Although this paper focused on the stabilization of the more common type of co-simulation, i.e. the explicit modular stepping, further performance gains are expected with more advanced types of co-simulation, for instance by combining the stabilization method with variable macro-step size heuristics or implicit (iterative) stepping, since these currently seldom used techniques are now enabled by the FMI specifications.

## Acknowledgments

This work was in part supported by DGCIS in the ITEA2 MODRIO (11004) project under contract number 12.2.93.0126.

## References

- [1] Akers A., Gassman M., Smith R., Hydraulic Power System Analysis, CRC Press, 2006.
- [2] Ascher U., Petzold L., Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, Society for Industrial and Applied Mathematics, Philadelphia, 1997.
- [3] Arnold M., Clauss C., Schierz T., Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-simulation V2.0, The Archive of Mechanical Engineering, Vol. LX, No 1, 2013.
- [4] Arnold M., Numerical stabilization of co-simulation techniques, the ODE case, Working document MODELISAR, sWP 200-203, September 5, 2011.
- [5] Modelica Association Project "FMI", Functional Mock-up Interface for Model Exchange and Co-simulation 2.0 Release Candidate 1, October 18, 2013, available at [https://svn.modelica.org/fmi/branches/public/specifications/FMI\\_for\\_ModelExchange\\_and\\_CoSimulation\\_v2.0\\_RC1.pdf](https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_and_CoSimulation_v2.0_RC1.pdf)
- [6] Franklin G.F., Powell J.D., Workman M.L., Digital Control of Dynamic Systems, Addison Wesley, 1997.
- [7] Kübler R., Schiehlen W., Two Methods of Simulator Coupling, Mathematical and Computer Modelling of Dynamical Systems, Vol. 6, No. 2, 2000.
- [8] Viel A., Strong Coupling of Modelica System-Level Models with Detailed CFD Models for Transient Simulation of Hydraulic Components in their Surrounding Environment, 8th International Modelica Conference, Dresden, March 2011.





# Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using FMI

Abir Ben Khaled<sup>1</sup> Laurent Duval<sup>1</sup> Mongi Ben Gaid<sup>1</sup> Daniel Simon<sup>2</sup>

<sup>1</sup>IFP Energies nouvelles, 1 et 4 avenue de Bois-Préau, 92852 Rueil-Malmaison, France

<sup>2</sup>INRIA and LIRMM - DEMAR team, 95 rue de la Galéra, 34090 Montpellier, France

abir.ben-khaled@ifpen.fr laurent.duval@ifpen.fr

mongi.ben-gaid@ifpen.fr daniel.simon@inria.fr

## Abstract

The growing complexity of systems, together with increasing available parallelism provided by multi-core chips, calls for the parallelization of simulation. Simulation speed-ups are expected from co-simulation and parallelization based on models splitting into loosely coupled sub-systems in the framework of Functional Mockup Interface (FMI). However, slackened synchronization between the sub-models and associated solvers running in parallel introduces integration errors, which must be kept inside predefined bounds. In this paper, context-based extrapolation is investigated to improve the trade-off between integration speed-ups, needing large communication steps, and simulation precision, needing frequent updates for the models inputs. An internal combustion engine, based on FMI for model exchange, is used to assess the parallelization methodology.

*Keywords:* FMI; parallel simulation; signal processing; polynomial extrapolation; real-time; context-based decision

## 1 Introduction

During the design process of complex systems, such as in automotive, simulation is proven to be an indisputable step between concept design and prototype validation. Realistic simulations allow for the preliminary evaluation, tuning and possibly redesign of proposed solutions ahead of implementation, thus lowering the risks. To be confident in the result, building such simulations requires high-fidelity models both for the components and for their interaction.

Currently, building high-fidelity system-level models of cyber-physical systems in general and automo-

tive cars in particular, is a challenging duty. One problem is the diversity of models, designed for different environments, provided by various multi-disciplinary teams. Distinctive environments are preferred for a specific use due to specific strengths (modeling language, libraries, solvers, cost...). The FMI specification has been proposed to improve this issue [1].

However, the simulation of high-fidelity models is time consuming, and reaching real-time constraints is out of the capabilities of single-threaded simulations running on single cores. Simulation speed-ups are needed, in particular by splitting the systems into sub-models to be executed in parallel on currently available multi-core chips.

Unfortunately most of the existing simulation software are currently unable to exploit multi-core platforms, as they rely on sequential Ordinary Differential Equations (ODE) and Differential Algebraic Equations (DAE) solvers. The co-simulation approaches can provide significant improvements by allowing to simulate together models coming from different areas, and to validate both the individual behaviors and their interaction [2]. The simulators may be exported from original authoring tools as Functional Mock-up Units (FMUs), and then imported in a co-simulation environment. Hence, they cooperate at run-time, thanks to the FMI definitions of their interfaces, and to the master algorithms of these environments.

The “FMI for model exchange” framework allows for solving independently the sub-models using custom solvers. In this context, several methods have been already proposed to perform real-time distributed simulation of complex physical models. For example, in [3], the study focused on the case of fixed-step solvers. Then, in [4], the study was extended to handle the case of variable time-step solvers.

However, accounting for the dependencies between

the sub-models needs to synchronize them at some time intervals. Certainly, this synchronization avoids the propagation of numerical errors in the simulation results and guarantees their correctness. Unfortunately, these synchronization constraints also lead to waiting periods and idle time of some processors. Consequently it decreases the potential efficiency of the threaded parallelism existing in multi-core platforms.

To overcome this limitation, and to more efficiently exploit the available parallelism, the dependencies constraints should be relaxed as far as possible while keeping accumulated errors under control. In a first step, this can be performed by a well done system decomposition that minimizes the dependencies between the sub-models. For example, a method was proposed in [5] for distributed simulation using a technology based on bilateral delay lines called transmission line modeling (TLM), where the decoupling point is chosen when the variables change slowly and the time-step of the solver is relatively small.

Unfortunately, most often perfect decoupling cannot be reached and data dependencies still exist between parallel blocks. Some synchronization between them must be kept tight through small communication steps between models, which prevents the variable time-step solvers to reach large integration steps.

It is proposed that the synchronization steps can be stretched out with limited deterioration of the simulation precision, thanks to a well-suited, albeit simple, context-based polynomial extrapolation of the exchanged data beyond the synchronization points between sub-models.

This paper is organized as follows. First, a formal model of a hybrid dynamical system is given and a model of the integration errors due to slack synchronization is sketched in Section 2. The background on prediction and polynomial prediction algorithms are developed in Section 3. The principles for context-based extrapolation, to cope with the hybrid nature of the models, are exposed in Section 4. The methodology is assessed in Section 5 using the model of an internal combustion engine.

## 2 Motivation for extrapolation

### 2.1 Model formalization

Consider a hybrid dynamical system  $\Sigma$  described by a set of nonlinear differential equations:

$$\begin{aligned}\dot{\mathbf{X}} &= \mathbf{f}(t, \mathbf{X}, \mathbf{D}, \mathbf{U}) \quad \text{for } t_n \leq t < t_{n+1}, \\ \mathbf{Y} &= \mathbf{g}(t, \mathbf{X}, \mathbf{D}, \mathbf{U}),\end{aligned}$$

where  $\mathbf{X} \in \mathbb{R}^{n_x}$  is the continuous state vector,  $\mathbf{D} \in \mathbb{R}^{n_d}$  is the discrete state vector,  $\mathbf{U} \in \mathbb{R}^{n_u}$  is the input vector,  $\mathbf{Y} \in \mathbb{R}^{n_y}$  is the output vector and  $t \in \mathbb{R}^+$  is the time.

The sequence  $(t_n)_{n \geq 0}$  of strictly increasing time instants represents discontinuity points called “state events”, which are the roots of the equation

$$h(t, \mathbf{X}, \mathbf{D}, \mathbf{U}) = 0.$$

The function  $h$  is usually called zero-crossing function or event indicator. It is used for *event detection* and *location* [6].

At each time instant  $t_n$ , a new continuous state vector can be computed as a result of the *event handler*

$$\mathbf{X}(t_n) = \mathbf{I}(t_n, \mathbf{X}, \mathbf{D}, \mathbf{U}),$$

and a new discrete state vector can be computed as a result of discrete state update

$$\mathbf{D}(t_n) = \mathbf{J}(t_{n-1}, \mathbf{X}, \mathbf{D}, \mathbf{U}).$$

If no discontinuity affects a component of  $\mathbf{X}(t_n)$ , the right limit of this component will be equal to its value at  $t_n$ .

It is assumed that  $\Sigma$  is well posed in the sense that a unique solution exists for each admissible initial conditions  $\mathbf{X}(t_0)$  and  $\mathbf{D}(t_0)$  and that consequently  $\mathbf{X}$ ,  $\mathbf{D}$ ,  $\mathbf{U}$ , and  $\mathbf{Y}$  are piece-wise continuous functions in  $[t_n, t_{n+1}]$ .

To execute the system in parallel, the model must be split into several sub-models. Assume for simplicity, that the system is decomposed into two subsystems as in Figure 1. Our approach generalizes to any decomposition into  $N$  blocks of system  $\Sigma$ .

Therefore, the system can be written as:

$$\begin{cases} \dot{\mathbf{X}}_1 = \mathbf{f}_1(\mathbf{X}_1, \mathbf{X}_2, \mathbf{D}_1, \mathbf{U}_1) \\ \mathbf{Y}_1 = \mathbf{g}_1(\mathbf{X}_1, \mathbf{X}_2, \mathbf{D}_1, \mathbf{U}_1) \end{cases} \quad \text{and} \quad \begin{cases} \dot{\mathbf{X}}_2 = \mathbf{f}_2(\mathbf{X}_1, \mathbf{X}_2, \mathbf{D}_2, \mathbf{U}_2) \\ \mathbf{Y}_2 = \mathbf{g}_2(\mathbf{X}_1, \mathbf{X}_2, \mathbf{D}_2, \mathbf{U}_2) \end{cases}$$

with  $\mathbf{X} = [\mathbf{X}_1 \ \mathbf{X}_2]^T$  and  $\mathbf{D} = [\mathbf{D}_1 \ \mathbf{D}_2]^T$ , where  $^T$  denotes the matrix transpose.

Here,  $\mathbf{U}_1$  are the inputs needed for  $\Sigma_1$  and  $\mathbf{U}_2$  are the inputs needed for  $\Sigma_2$ . In other words,  $\mathbf{U}_1 \cup \mathbf{U}_2 = \mathbf{U}$

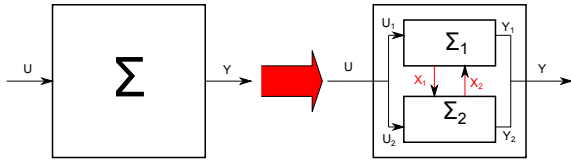
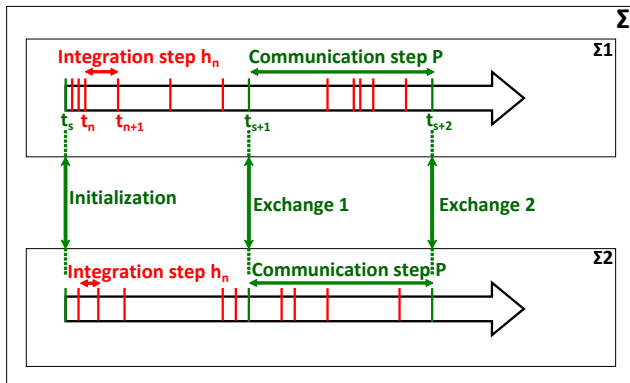


Figure 1: System splitting for parallelization.

and  $U_1 \cap U_2$  can be an empty set or not according to the achieved decoupling.

In the same way,  $Y_1$  are the outputs produced by  $\Sigma_1$  and  $Y_2$  are the outputs produced by  $\Sigma_2$ . In other words,  $Y_1 \cup Y_2 = Y$  and  $Y_1 \cap Y_2 = \emptyset$ .

To perform the numerical integration of the whole multivariable system, each of these simulators needs to exchange, at communication points, the data needed by the others (see Figure 2). To speed up the integration, the parallel branches must be as independent as possible, so that they are synchronized at a rate  $P$  by far slower than their internal integration step  $h_n$  ( $P \gg h_n$ ). Therefore, between communication points, each simulator integrates at its own rate (assuming a variable step solver), and considers that the data incoming from others simulators is hold as constant.


 Figure 2:  $\Sigma$  split into  $\Sigma_1$  and  $\Sigma_2$  for parallel simulation.

It is likely that large communication intervals allow to speed up the numerical integration, but may result in integration errors and poor confidence in the final result. Modeling the errors induced by slack synchronization is a first step to find effective directions to improve the trade-offs between integration speed and accuracy.

## 2.2 Integration errors and parallelism

To compute the next state value  $X_i(t_{n+1})$ ,  $i = 1, 2$  (see Figure 3), the numerical solver needs at least the values of  $X_i(t_n)$  and  $\dot{X}_i(t_n) = f_i(X(t_n))$  (e.g. for Euler

integration). The inputs and discrete states are omitted for clarity.

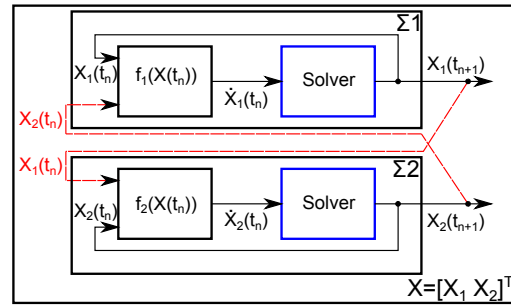


Figure 3: System's internal composition.

When computing  $\dot{X}_1(t_n) = f_1(X(t_n))$ , the value of the local variable  $X_1(t_n)$  is always available. This is not the case for  $X_2(t_n)$ , which is computed in a parallel branch. In fact,  $X_2$  is only available in branch 1 at synchronization with interval  $P$ , which is larger than the integration step  $h_n$ . In other words,  $X_2(t_n)$  is available only when the time  $t_n$  corresponds with a synchronization point  $t_s$  (see Figure 2), otherwise its estimated value is the one transmitted at the previous synchronization point. Let us evaluate the evolution of integration errors due to slack synchronization between the parallel branches when computing  $\dot{X}_1(t_n) = f_1(X(t_n))$ . The analysis on  $\Sigma_1$  remains valid for  $\Sigma_2$ .

The influence of using a delayed value of  $X_2$  in  $f_1(\cdot)$  (respectively  $X_1$  in  $f_2(\cdot)$ ) is due to the lack of updated data during a delay  $\tau$ , represented by the difference between the current integration time  $t_n$  and the last synchronization time  $t_s$  as

$$\tau = t_n - t_s \quad (1)$$

with

$$t_s = P \left\lfloor \frac{t_n}{P} \right\rfloor$$

therefore

$$t_s = \begin{cases} lP & \text{when } t_n = lP \quad l \in \mathbb{N}^* \\ (l-1)P & \text{when } t_n < lP \quad l \in \mathbb{N}^* \end{cases}$$

leading to

$$\begin{cases} \tau = 0 & \text{when } t_n = t_s \\ \tau > 0 & \text{when } t_n > t_s \end{cases}$$

Therefore, the induced error at  $t_{n+1}$  in the subsystem  $\Sigma_1$ , denoted  $E_1(t_{n+1})$ , is the difference between  $X_1(t_{n+1})$  for the unsplit model (2) and  $\tilde{X}_1(t_{n+1})$  for the split model (3):

$$\mathbf{X}_1(t_{k+1}) = \mathbf{X}_1(t_k) + h_k \mathbf{f}_1(\mathbf{X}_1(t_k), \mathbf{X}_2(t_k)), \quad k \in \{0, \dots, n\} \quad (2)$$

$$\tilde{\mathbf{X}}_1(t_{k+1}) = \begin{cases} \mathbf{X}_1(t_{k+1}) & k=0 \\ \tilde{\mathbf{X}}_1(t_k) + h_k \mathbf{f}_1(\tilde{\mathbf{X}}_1(t_k), \tilde{\mathbf{X}}_2(t_k - \tau)) & k \geq 1 \end{cases} \quad (3)$$

In other words,

$$\begin{aligned} \mathbf{E}_1(t_{n+1}) &= \sum_{k=0}^n \mathbf{E}_1(t_k) \\ &+ h_n [\mathbf{f}_1(\mathbf{X}_1(t_n), \mathbf{X}_2(t_n)) - \mathbf{f}_1(\tilde{\mathbf{X}}_1(t_n), \tilde{\mathbf{X}}_2(t_n - \tau))] \\ &= \mathbf{E}_{1,p}(t_n) + \mathbf{E}_{1,c}(t_{n+1}) \end{aligned}$$

where

$$\begin{aligned} \mathbf{E}_{1,c}(t_{n+1}) &= h_n [\mathbf{f}_1(\mathbf{X}_1(t_n), \mathbf{X}_2(t_n)) - \mathbf{f}_1(\tilde{\mathbf{X}}_1(t_n), \tilde{\mathbf{X}}_2(t_n - \tau))] \\ \mathbf{E}_{1,p}(t_n) &= \sum_{k=0}^n \mathbf{E}_1(t_k) \end{aligned} \quad (4)$$

Here  $\mathbf{E}_{1,c}(t_{n+1})$  is the current error generated at  $t_{n+1}$  whatever a synchronization or not. So, the global decoupling error  $\mathbf{E}_1(t_{n+1})$  is the result of the accumulation of past errors  $\mathbf{E}_{1,p}(t_n)$  and the current error  $\mathbf{E}_{1,c}(t_{n+1})$ . As a conclusion, to achieve a correct result, two conditions must be met for the current (local) error and the global error:

- $|\mathbf{E}_{1,c}(t_{n+1})| < \epsilon_{\text{loc}}$ : allowed local error
- $|\mathbf{E}_1(t_{n+1})| < \epsilon_{\text{glo}}$ : allowed global error

These conditions can be satisfied by acting on some parameters. Indeed, in (4), the delay error depends on the integration steps  $h_n$  and on the delay  $\tau$ . The integration step  $h_n$  is already adapted following the numerical solver strategy and the user-defined solver tolerance. The delay  $\tau$ , however, depends on the last synchronization time  $t_s$ , which is function of the synchronization period  $P$ .

The delay induced error tends to zero when the delay  $\tau$  tends to zero, which means that the delay error can be eliminated with the synchronization interval set equal to the integration steps. In other words, all the parallel subsystems should be integrated at the same adaptive rate (in the case of adaptive synchronization period), or with same fixed time-step. These two assumptions are very restrictive, as they force to choose a global adequate time-step regardless the discontinuities and the stiffness of the sub-systems. Compared with the single-threaded simulation, the only possible speed-ups during a parallel execution would be brought by the brute force computation power of the multicore machine, reduced by the parallelization cost.

Therefore, considering a split model and a parallel execution, a trade-off must be found between acceptable simulation errors, thanks to tight enough synchronization, and simulation speed-ups thanks to decoupling between sub-models.

To add a degree of freedom to this trade-off achievement, we propose to extrapolate model inputs to compensate the stretching out of the communication steps between sub-models. Note that in this first approach of the polynomial extrapolation, the synchronization interval is considered as constant. Future enhancements will consider communication step size control, for which the error analysis and estimation can be inspired by [7]. Extrapolation is sensitive for different reasons:

- prediction should be efficient: causal, sufficiently fast and reliable;
- there exist no universal prediction scheme, efficient with every signal;
- polynomial prediction may fail in stiff cases [8] (cf. Section 4 for details).

We choose to base our extrapolation on polynomial prediction, which allows fast and causal calculations. The rationale is that, in this situation, the computing cost of a low-order polynomial predictor would be by far smaller than the extra model computations needed by shorten communication steps. Since such predictions would be accurate neither for any signal (for instance, blocky versus smooth signals) nor any signal behavior (slow variations versus steep onsets), we borrow a context-based approach, common with lossless image coders [9], such as GIF or PNG formats. The general aim of these image coders is to predict a pixel value based on a pattern of causal neighboring pixels. Compression is obtained when the prediction residues possess smaller intensity values, and more generally a better distribution (concentrated around close-to-zero values) than the pixels in the original image. They may thus be coded on smaller “bytes”, using entropy coding techniques. In images, one distinguishes basic “objects” such as smooth-intensity varying regions, or edges with different orientations. Based on simple calculation of the prediction pattern pixels, different contexts are inferred (e.g. flat, smooth,  $+45^\circ$  or  $-45^\circ$  edges, etc.). Look-up table predictors are then used, depending on the context.

In the proposed approach, we build a heuristic table of contexts (in Section 4) based on a short frame of past samples, and affect a pre-determined polynomial

predictor to obtain a context-dependent extrapolated value. We now review the principles of extrapolation.

### 3 Causal polynomial prediction

#### 3.1 Background on prediction

This section is dedicated to a peculiar instance of discrete time series, or signal, forecasting. The neighboring topics of prediction or extrapolation represent a large body of knowledge in signal processing [10], econometrics [11] or control [12].

In the present case, we consider a real-valued, regularly sampled signal  $u$ , with period  $P$ , known at synchronization or communication intervals. Prediction in general assumes the knowledge of signal formation models. Since very little is assumed on the signal's dynamics (no behavioral/explicit model is available, periodicity and regularity are unknown), and as we operate under real-time conditions, implying strong causality, only a tiny fraction of time series methods are practically applicable. Zeroth-order hold or nearest-neighbor extrapolation is probably the most natural, the less hypothetical, and the less computationally expensive forecasting method. It consists in using the latest known sample as the predicted value. It possesses small (cumulative) errors when the time series is relatively flat or its sampling rate is sufficiently high, with respect to the signal's dynamics. In other words, it is efficient when the time series is sampled fast enough to ensure small variations between two consecutive sampling times. However, it indirectly leads to under-sampling related disturbances, that affect the signal content. They appear as quantization-like noise, offset or peak flattening.

In our co-simulation framework, communication intervals are not chosen arbitrarily small for computational efficiency. Thus, the slow variation of inputs and outputs cannot be ensured in practice. Hence, borrowing additional samples from the past known data and using higher-order extrapolation methods could be beneficial, provided a trade-off of cost and error is met. Different forecast methods of various fidelity and complexity may be efficiently evaluated. We focus here on polynomial methods, for their simplicity and ease of implementation, following initial works in [13, Chapter 16].

#### 3.2 Notations

We denote by  $P_{(\delta,\lambda)}$  the least-squares polynomial predictor of degree  $\delta \in \mathbb{N}$  and prediction length  $\lambda \in \mathbb{N}^*$ .

The prediction length  $\lambda$  represents the number of past samples required for each prediction, performed in the least-squares sense [14, p. 227 sq.]. For convenience, we use a 0-last-sample-index convention: we re-index the frame of the  $\lambda$  past samples such that the last known sample is indexed by 0. Computations for the prediction at relative time  $\tau$  (loosely denoted by  $u(\tau)$ ), defined in (1), thus require past samples  $\{u_{1-\lambda}, u_{2-\lambda}, \dots, u_0\}$ . We first recall principles and formulas for a standard least-squares, degree-two or parabolic prediction. The general equations are derived next.

#### 3.3 Polynomial prediction of degree $\delta = 2$

We look for the best fitting parabola, i.e. with degree  $\delta = 2$ ,  $u(t) = a_\delta + a_{\delta-1}t + a_{\delta-2}t^2$  to approximate the set of discrete samples  $\{u_{1-\lambda}, u_{2-\lambda}, \dots, u_0\}$ . The prediction polynomial  $P_{(2,\lambda)}$  is defined by the vector of polynomial coefficients  $\mathbf{a} = [a_2, a_1, a_0]^T$ . They are determined, in the least-squares sense [15], by minimizing the squared or quadratic, error:

$$e(\mathbf{a}) = \sum_{l=1-\lambda}^0 (u_l - (a_2 + a_1 l + a_0 l^2))^2.$$

Note that the  $l$  indices here are non-positive, between  $1 - \lambda$  and 0. The minimum error is obtained by solving the following system of equations (zeroing the derivatives with respect to each of the free variables  $a_i$ ):

$$\forall i \in \{0, 1, 2\}, \quad \frac{\partial e(\mathbf{a})}{\partial a_i} = 0$$

namely:

$$\begin{cases} \sum_{l=1-\lambda}^0 l^0 (u_l - (a_2 l^0 + a_1 l^1 + a_0 l^2)) = 0, \\ \sum_{l=1-\lambda}^0 l^1 (u_l - (a_2 l^0 + a_1 l^1 + a_0 l^2)) = 0, \\ \sum_{l=1-\lambda}^0 l^2 (u_l - (a_2 l^0 + a_1 l^1 + a_0 l^2)) = 0. \end{cases} \quad (5)$$

The system in (5) may be rewritten as:

$$\begin{cases} \sum_{l=1-\lambda}^0 u_l = a_2 \sum_{l=1-\lambda}^0 l^0 + a_1 \sum_{l=1-\lambda}^0 l^1 + a_0 \sum_{l=1-\lambda}^0 l^2, \\ \sum_{l=1-\lambda}^0 l u_l = a_2 \sum_{l=1-\lambda}^0 l^1 + a_1 \sum_{l=1-\lambda}^0 l^2 + a_0 \sum_{l=1-\lambda}^0 l^3, \\ \sum_{l=1-\lambda}^0 l^2 u_l = a_2 \sum_{l=1-\lambda}^0 l^2 + a_1 \sum_{l=1-\lambda}^0 l^3 + a_0 \sum_{l=1-\lambda}^0 l^4. \end{cases}$$

Let  $m^d = \sum_{l=0}^{\lambda-1} l^{\delta-d} u_{-l}$  (here the indices  $l$  are positive) denote the  $(\delta-d)$ -th moment of the frame  $u_i$ , and  $\mathbf{m}$  the vector of moments  $[m^2, -m^1, m^0]^T$ . We express the sums of integer powers by  $\Sigma_\lambda^d = \sum_{i=0}^{\lambda-1} i^d$ . Closed-form expressions exist for  $\Sigma_\lambda^d$ , involving Bernoulli sequences [16]. For instance:

- $\Sigma_\lambda^0 = \lambda$ ,
- $\Sigma_\lambda^1 = (\lambda-1)\lambda/2$ ,
- $\Sigma_\lambda^2 = (\lambda-1)\lambda(2\lambda-1)/6$ ,
- $\Sigma_\lambda^3 = (\lambda-1)^2\lambda^2/4$ ,
- $\Sigma_\lambda^4 = (\lambda-1)\lambda(2\lambda-1)(3\lambda^2-3\lambda-1)/30$ .

We now form the matrix  $\mathbf{Z}_{(2,\lambda)}$  of sums of powers (depending on  $\delta = 2$  and  $\lambda$ ):

$$\mathbf{Z}_{(2,\lambda)} = \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \Sigma_\lambda^2 \\ -\Sigma_\lambda^1 & \Sigma_\lambda^2 & -\Sigma_\lambda^3 \\ \Sigma_\lambda^2 & -\Sigma_\lambda^3 & \Sigma_\lambda^4 \end{bmatrix}.$$

The system in (5) rewrites:

$$\begin{bmatrix} m^2 \\ -m^1 \\ m^0 \end{bmatrix} = \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \Sigma_\lambda^2 \\ -\Sigma_\lambda^1 & \Sigma_\lambda^2 & -\Sigma_\lambda^3 \\ \Sigma_\lambda^2 & -\Sigma_\lambda^3 & \Sigma_\lambda^4 \end{bmatrix} \times \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

or  $\mathbf{m} = \mathbf{Z}_{(2,\lambda)} \times \mathbf{a}$ . Now we want to find the value predicted by  $P_{(2,\lambda)}$  at time  $\tau$ . Let  $\boldsymbol{\tau}_2 = [1, \tau, \tau^2]^T$  be a vector of  $\tau$  powers. Then  $u(\tau)$  is equal to  $a_2 + a_1\tau + a_0\tau^2 = \boldsymbol{\tau}_2^T \times \mathbf{a}$ . Finally,  $\mathbf{Z}_{(2,\lambda)}$  is always invertible, provided that  $\lambda > \delta$ . Its inverse is denoted  $\mathbf{Z}_{(-2,\lambda)}$ . It thus does not need to be updated in real-time. It may be computed off-line, numerically or even symbolically. Hence:

$$u(\tau) = (\boldsymbol{\tau}_2^T \times \mathbf{Z}_{(-2,\lambda)}) \times \mathbf{m}.$$

The vector  $\boldsymbol{\tau}_2$  and  $\mathbf{Z}_{(-2,\lambda)}$  are fixed, and the product  $\boldsymbol{\tau}_2^T \times \mathbf{Z}_{(-2,\lambda)}$  may be stored at once. Thus, for each prediction, the only computations are the update of the vector  $\mathbf{m}$  and his product with the aforementioned stored matrix. It thus enables look-up-table-based predictions, which helps to reduce propagation errors in matrix computations.

### 3.4 General formulas

Inferring from the previous example, we easily get a more generic extrapolation pattern in its matrix form:

$$u(\tau) = \begin{bmatrix} 1 & \tau & \dots & \tau^\delta \end{bmatrix} \times \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \dots & (-1)^\delta \Sigma_\lambda^\delta \\ -\Sigma_\lambda^1 & \dots & \dots & \vdots \\ \vdots & \dots & \dots & \vdots \\ (-1)^\delta \Sigma_\lambda^\delta & \dots & \dots & \Sigma_\lambda^{2\delta} \end{bmatrix}^{-1} \times \begin{bmatrix} m^\delta \\ -m^{\delta-1} \\ \vdots \\ (-1)^\delta m^0 \end{bmatrix}.$$

Note  $\boldsymbol{\tau}_\delta = [1, \tau, \dots, \tau^\delta]^T$ , then:

$$u(\tau) = \boldsymbol{\tau}_\delta^T \mathbf{Z}_{(-\delta,\lambda)} \mathbf{m}.$$

As in the previous case, only  $\mathbf{m}$  and one matrix product need be computed in real-time. When  $\delta = 0$ , one easily sees that:

$$u(\tau) = \frac{m^0}{\Sigma_\lambda^0} = \frac{u_{1-\lambda} + \dots + u_0}{\lambda},$$

that is, the running average of past frame values, reducing to the zeroth-order hold when  $\lambda = 1$ . Although the matrix formulation is convenient, actual computation does not require true matrix calculus, especially for small degrees  $\delta$ . For instance,  $P_{(1,3)}$  yields the simple estimator form:  $u(\tau) = \frac{\tau}{2}(u_0 - u_{-2}) + \frac{1}{6}(5u_0 + 2u_{-1} - u_{-2})$ .

## 4 Context-based extrapolation

Actual complex systems usually present nonlinearities and discontinuities, so that it is hard to predict their future behavior from past observations. Moreover the considered models are generated using the FMI for model exchange framework, which does not provide the inputs' derivatives (conversely with the FMI for co-simulation architecture). Hence the previously described polynomial prediction cannot correctly extrapolate along all the system trajectories.

For example, [17] studies a method based on a sequential implementation of continuous dynamical systems that uses a constant, linear or quadratic extrapolation and a linear interpolation to improve the accuracy of the modular time integration. The study shows that the method is successful for non-stiff systems and it

fails for the stiff case. The context-based extrapolation is then performed to account for steps, stiffness, discontinuities or weird behavior, and use adapted extrapolation to limit excessively wrong prediction.

Keeping with the previous 0-last-sample-index convention, and for the sake of simplicity, we first define a measure of variation based on the last three samples:  $d_0 = u_0 - u_{-1}$  and  $d_1 = u_{-1} - u_{-2}$ , the last and previous differences. Their absolute values are compared with two thresholds,  $\gamma_0$  and  $\gamma_{-1}$ , respectively. We then define three complementary conditions:

- $O$  if  $|d_i| = 0$ ;
- $C_i$  if  $0 < |d_i| \leq \gamma_i$ ;
- $\overline{C}_i$  if  $|d_i| > \gamma_i$ ;

We can now define the six-context Table 1, and examples for their associated heuristic polynomial predictors. The six contexts form a partition, i.e. they are mutually exclusive, and cover all possible options for a hybrid dynamical system. They are illustrated in Figure 4. Their names represent their behavior. For instance, the flat context addresses steady signals, for which a mere zeroth-order hold suffices, hence  $P_{(0,1)}$ . The calm context represents a sufficiently sampled situation, where value increments over time remain below fixed thresholds. In this case, the signal is relatively regular, and could be approximated by a quadratic polynomial, for instance  $P_{(2,5)}$ . For the “flat” and “jump” contexts, there is an additional procedure which consists in resetting the extrapolation to prevent inaccurate prediction. For example, when context 1 is chosen just after context 5, the quadratic extrapolation  $P_{(2,5)}$  requires 5 valid samples, whereas the last 3 only are relevant.

Our two-threshold is relatively simple. Hence, the choice of the thresholds  $\gamma_0$  and  $\gamma_{-1}$ , is potentially crucial. For instance, fixed values may reveal inefficient under important amplitude or scale variation of signal. Hence, we have chosen here to compute them, in a running manner, on the past frame  $\{u_{1-\omega}, \dots, u_{-3}\}$ . With excessively low thresholds, high-order extrapolations would be rarely chosen, loosing the benefits of predictions. Too high thresholds would in contrast suffer from any unexpected jump or noise. As the contexts are based on backward derivatives, we have used in the simulations presented here the mid-range statistical estimator of their absolute values. This amounts to set:  $\gamma_0 = \gamma_{-1} = \frac{1}{2} \max_{i \in [1-\omega, \dots, -3]} (|u_i - u_{i+1}|)$ .

Table 1: Summary of the six-context Table.

n(ame)	#	$ d_{-2} $	$ d_{-1} $	$d_{-2} \cdot d_{-1}$	$(\delta, \lambda)$
f(lat)	0	$O$	$O$	$O$	(0, 1)
c(alm)	1	$C_1$	$\overline{C}_2$	any	(2, 5)
m(ove)	2	$\overline{C}_1$	$\overline{C}_2$	any	(0, 1)
r(est)	3	$\overline{C}_1$	$C_2$	any	(0, 2)
t(ake)	4	$\overline{C}_1$	$\overline{C}_2$	$> 0$	(1, 3)
j(ump)	5	$\overline{C}_1$	$\overline{C}_2$	$< 0$	(0, 1)

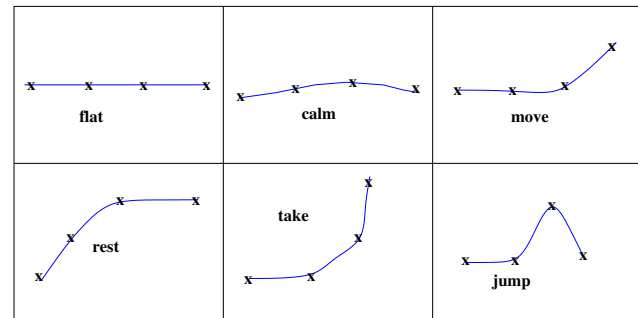


Figure 4: Illustration for context table in Table 1.

## 5 Case study

### 5.1 Engine simulator

In this study, a Spark Ignition (SI) RENAULT F4RT engine has been modeled. It is a four-cylinder in-line Port Fuel Injector (PFI) engine in which the engine displacement is 2000 cm<sup>3</sup>. The air path (AP) consists in a turbocharger with a mono-scroll turbine controlled by a waste-gate, an intake throttle and a downstream-compressor heat exchanger.

The engine model was developed using the ModEngine library [18]. ModEngine is a Modelica [19] library that allows the modeling of a complete engine with diesel and gasoline combustion models. Requirements for the ModEngine library were derived from the existing IFP-Engine AMESim<sup>1</sup> library. ModEngine contains more than 250 sub-models. It has been developed to allow the simulation of a complete virtual engine using a time-scale related to fractions of the crankshaft angle. A variety of elements are available to build representative models for engine components, such as turbocharger, wastegate, gasoline or Diesel injectors, valve, air path, EGR loop etc. ModEngine is currently functional in the Dymola tool<sup>2</sup>.

The engine model and the split parts were imported into xMOD model integration and virtual experimen-

<sup>1</sup>www.lmsintl.com/imagine-amesim-1-d-multi-domain-system-simulation

<sup>2</sup>www.3ds.com/products/catia/portfolio/dymola

tation tool [20], using the FMI export features of Dymola. The engine model has 118 state variables and 312 event indicators (of discontinuities).

## 5.2 Decomposition approach

The partitioning of the engine model is performed by separating the four-cylinder from the air path (AP), then by isolating the cylinders ( $C_i$ , for  $i \in [1, 2, 3, 4]$ ) from each other. This kind of splitting allows for the reduction of the number of events acting on each subsystem. In fact, the combustion phase raises most of the events, which are located in the firing cylinder. The solver can process them locally during the combustion cycle of the isolated cylinder, and then enlarge its integration time-step until the next cycle.

From a thermodynamic point of view, the cylinders are loosely coupled, but a mutual data exchange does still exist between them and the air path. The model is split into 5 components and governed by a basic controller denoted CTRL. It gathers 91 inputs and 98 outputs.

## 6 Tests and results

Tests are performed on a platform with 16GB RAM and 2 “Intel Xeon” processors, each running 8 cores at 3.1 GHz.

### 6.1 Reference simulations

The model validation is based on the observation of some quantities of interest as the intake and exhaust manifold pressures, air-fuel equivalence ratio and torque. These outputs are computed using LSODAR which is a variable time-step solver with a root-finding capability to detect the events occurring during the simulation. It has also the ability to adapt the integration method depending on the observed system stiffness.

The simulation reference  $Y_{\text{ref}}$  is built from the integration of the entire engine model, the solver tolerance (tol) being decreased until reaching stable results, which is reached for  $\text{tol} = 10^{-7}$  (at the cost of an unacceptable slow simulation speed).

Then, to explore the trade-offs between the simulation speed and precision, simulations are run with increasing values of the solver tolerance until reaching a desired relative integration error  $Er$ , defined by (6)

$$Er(\%) = \frac{100}{N} \cdot \sum_{i=0}^{N-1} \left( \left| \frac{Y_{\text{ref}}(i) - Y(i)}{Y_{\text{ref}}(i)} \right| \right) \quad (6)$$

with  $N$  the number of saved points during 1 s of simulation. Iterative runs showed that the relative error converge to a desired error ( $Er \leq 1\%$ ) for  $\text{tol} = 10^{-4}$ . The single thread simulation of the whole engine with LSODAR and  $\text{tol} = 10^{-4}$  provides the simulation execution time reference, to which the parallel versions are compared. When using the split model, each of its 5 components is assigned to a dedicated core and integrated by LSODAR with  $\text{tol} = 10^{-4}$ .

### 6.2 Effect of the context-based extrapolation on accuracy

To explore the effect of extrapolation on accuracy, the communication step has been set to  $250\mu\text{s}$  in a first set of experiments. This value has been chosen to provide acceptable results for the accuracy ( $Er \approx 1\%$ ), while being large enough to make extrapolation useful.

The tests show that performing only a fixed polynomial prediction (conventional first and second order extrapolation) on the engine model fails, with integration errors larger than for the reference simulation. This is due to the hybrid nature of the model, for which the extrapolation failures are caused by discontinuities, and also by sharp variations of some variables at specific instants. These cases totally waste the gain in precision due to successful extrapolation in the other parts of the state trajectories.

In contrast, using the context-based polynomial predictor, the outputs of the simulation are almost always closer to the reference trajectory than those computed when considering the inputs hold as constant (Figure 5).

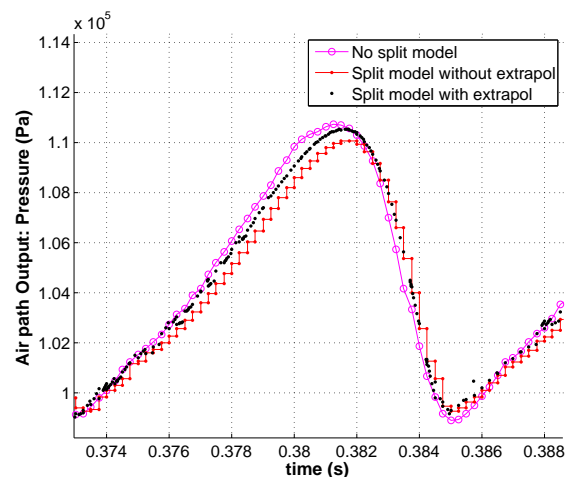


Figure 5: Airpath output: pressure.

Figure 6 shows that using context-based extrapola-



tion, the prediction step is discarded when there is a discontinuous behavior in the signal, and that the degree of the predictor is adapted according to the signal slope (Figure 6).

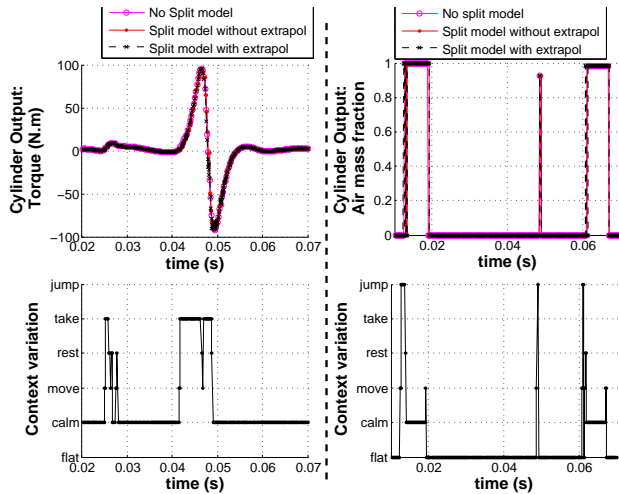


Figure 6: Context behavior during simulation.

The cumulative relative integration error on a long simulation run is computed in Table 2. It shows that the context-based extrapolation efficiently decreases this error for the chosen variables, for example by 63% for the temperature and by 72.5% for the fuel density.

Table 2: Relative integration error.

Outputs	Er(%)	Er(%)
	w/o extrapolation	w/ extrapolation
Pressure	0.499	0.304
Temperature	0.511	0.19
Air density	0.784	0.31
Fuel density	3.55	0.978
Burned gas density	4.99	3.47

### 6.3 Effect of the context-based extrapolation on simulation time

The ultimate objective of extrapolation is to decrease the simulation by stretching out the synchronization interval, while keeping the relative integration error  $Er$  inside predefined bounds. Indeed, widening the communication step from  $100\mu\text{s}$  to  $250\mu\text{s}$  without extrapolation (Figure 7) saves time but increases the error (e.g. 6.97% for the burned gas density and 340.5% for the fuel density).

Using the extrapolation for the  $250\mu\text{s}$  step fortunately decreases the relative error to values close to, or

below, those measured for the  $100\mu\text{s}$  step with frozen inputs.

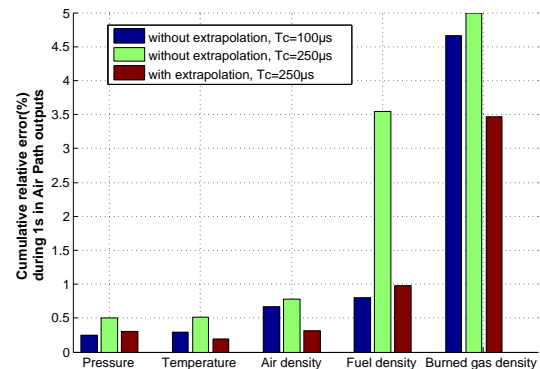


Figure 7: Cumulative relative error using different communication steps.

Table 3 shows the simulation speed-up compared with the single-threaded reference. First note that when splitting the model into 5 threads integrated in parallel on 5 cores, the speed-up is supra-linear w.r.t. the number of cores. Indeed, the containment of events detection and handling inside small sub-systems allows for solvers accelerations, enough to over-compensate the multi-threading costs. Secondly, it appears that combining the enlarged communication step and the context-based extrapolation, the 10% extra speed-up is reached without loss for the relative error. Even more surprising, using the extrapolation slightly speeds-up the simulation, possibly because the inputs shaped by the predictor enables a faster convergence of the solver step.

Table 3: Simulation speed-up.

Communication time	100µs		250µs	
	No	No	Yes	Yes
Speed-up	8.9	10.01	10.07	

## 7 Conclusion and future work

The aim of this work is to speed up the numerical integration of hybrid dynamical systems, eventually until reaching a real-time execution, while keeping the integration errors inside controlled bounds. The basic approach consists in splitting the system into sub-models, which are integrated in parallel. Using large synchronization intervals between the branches allows for numerical integration speed-ups. However, slack

synchronization intervals may generate integration errors in the final result.

In this paper, the errors caused by the slack synchronization are modeled, giving directions to find effective trade-offs between integration speed and accuracy. Then, an approach of stretching out the communication steps while keeping a predefined integration precision is proposed. Rather than using costly small integration and communication steps, it uses extrapolations of the behavior of the models over the synchronization intervals. Test results on a hybrid dynamical engine model, show that well chosen context-based extrapolation allows for an effective speed-up of the simulation with negligible computing overheads.

This work shows that properly-chosen context-based extrapolation, combined with model splitting and parallel integration, can potentially improve the speed/precision trade-off needed to reach real-time simulation. However, the accuracy could be widely improved by accessing on the input derivatives of the models. This is the case for the FMI for co-simulation, and it would be highly useful to also integrate this feature in the FMI for model exchange. Future works intend to improve the context-based extrapolation algorithm, to make it more subtly aware of data freshness and even more decrease the prediction induced integration errors. Another possibility is to process the input signals to separate them into simpler components, easier to predict with different predictors, and to cope with noise. When it comes to polynomials, wavelet pre-processors [21] could be useful, as they play an important role in polynomial model fitting.

## References

- [1] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The functional mockup interface for tool independent exchange of simulation models. In *8th Int. Modelica Conf.*, Germany, Mar. 2011.
- [2] M. Valasek. *Simulation Techniques for Applied Dynamics*, chapter Modeling, simulation and control of mechatronical systems. Volume 507 of Arnold and Schiehlen [8], 2008. Courses and Lectures.
- [3] C. Faure, M. Ben Gaïd, N. Pernet, M. Fremovici, G. Font, and G. Corde. Methods for real-time simulation of cyber-physical systems: application to automotive domain. In *IEEE Int. Wirel. Comm. Mobi. Comput. Conf. IWCMC'11*, pages 1105–1110, 2011.
- [4] A. Ben Khaled, M. Ben Gaïd, D. Simon, and G. Font. Multicore simulation of powertrains using weakly synchronized model partitioning. In *E-COSM'12 IFAC Workshop on Engine and Powertrain Control Simulation and Modeling*, Rueil-Malmaison, France, Oct. 2012.
- [5] M. Sjölund, R. Braun, P. Fritzson, and P. Krus. Towards efficient distributed simulation in Modelica using transmission line modeling. In *3rd Int. Workshop on Equation-Based Object-Oriented Modeling Languages and Tools EOOLT*, pages 71–80. Linköping Univ. Electronic Press, 2010.
- [6] F. Zhang, M. Yeddanapudi, and P. Mosterman. Zero-crossing location and detection algorithms for hybrid system simulation. In *Proc. 17th IFAC World Congress*, pages 7967–7972, Seoul, South Korea, Jul. 2008.
- [7] M. Arnold, C. Clauss, and T. Schierz. Error analysis and error estimates for co-simulation in FMI for model exchange and co-simulation V2.0. *Arch. Mech. Eng.*, LX(1):6–156, 2013.
- [8] M. Arnold and W. Schiehlen, editors. *Simulation Techniques for Applied Dynamics*, volume 507 of *CISM International Centre for Mechanical Sciences*. SpringerWienNewYork, 2008. Courses and Lectures.
- [9] M. J. Weinberger, G. Seroussi, and G. Sapiro. The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. *IEEE Trans. Image Process.*, 9(8):1309–1324, 2000.
- [10] N. Wiener. *Extrapolation, interpolation, and smoothing of stationary time series*. MIT Press, 1949.
- [11] R. G. Brown. *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall, 1962.
- [12] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Probability and Statistics. Wiley, 2008.
- [13] C. Faure. *Real-time simulation of physical models toward hardware-in-the-loop validation*. PhD thesis, Univ. Paris-Est, 2011.
- [14] R. W. Hamming. *Numerical methods for scientists and engineers*. Dover publications, 1973.
- [15] S. M. Stigler. Gauss and the invention of least squares. *Ann. Stat.*, 9(3):465–474, 1981.
- [16] G. F. C. de Bruyn and J. M. de Villiers. Formulas for  $1 + 2^p + 3^p + \dots + n^p$ . *Fibonacci Q.*, 32(3):271–276, 1994.
- [17] M. Arnold. *Simulation Techniques for Applied Dynamics*, chapter Numerical methods for simulation in applied dynamics. Volume 507 of Arnold and Schiehlen [8], 2008. Courses and Lectures.
- [18] Z. Benjelloun-Touimi, M. Ben Gaïd, J. Bohbot, A. Dutoya, H. Hadj-Amor, P. Moulin, H. Saafi, and N. Pernet. From physical modeling to real-time simulation: Feedback on the use of modelica in the engine control development toolchain. In *8th Int. Modelica Conf.*, Dresden, Germany, 2011. Linköping Univ. Electronic Press.
- [19] P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. Wiley.com, 2010.
- [20] M. Ben Gaïd, G. Corde, A. Chasse, B. Léty, R. De La Rubia, and M. Ould Abdellahi. Heterogeneous model integration and virtual experimentation using xMOD : Application to hybrid powertrain design and validation. In *7th EUROSIM Congress on Modeling and Simulation*, Prague, Czech Republic, Sep. 2010.
- [21] C. Chaux, J.-C. Pesquet, and L. Duval. Noise covariance properties in dual-tree wavelet decompositions. *IEEE Trans. Inform. Theory*, 53(12):4680–4700, Dec. 2007.

# Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems

Himanshu Neema<sup>1</sup>

Jesse Gohl<sup>2</sup>

Zsolt Lattmann<sup>1</sup>

Janos Sztipanovits<sup>1</sup>

Gabor Karsai<sup>1</sup>

Sandeep Neema<sup>1</sup>

Ted Bapty<sup>1</sup>

John Batteh<sup>2</sup>

Hubertus Tummescheit<sup>2</sup>

Chandrasekar Sureshkumar<sup>2</sup>

<sup>1</sup>Institute for Software Integrated Systems, Vanderbilt University  
1025 16th Avenue South, Suite 102, Nashville, TN 37212, USA

<sup>2</sup>Modelon, Inc.

2389 Main Street, Glastonbury, CT 06033, USA

himanshu@isis.vanderbilt.edu jesse.gohl@modelon.com lattmann@isis.vanderbilt.edu sz-  
ipaj@isis.vanderbilt.edu gabor@isis.vanderbilt.edu sandeep@isis.vanderbilt.edu bap-

ty@isis.vanderbilt.edu john.batteh@modelon.com hubertus.tummescheit@modelon.com chandra-  
sekar.sureshkumar@modelon.com

## Abstract

Virtual evaluation of complex Cyber-Physical Systems (CPS) [1] with a number of tightly integrated domains such as physical, mechanical, electrical, thermal, cyber, etc. demand the use of heterogeneous simulation environments. Our previous effort with C2 Wind Tunnel (C2WT) [2] [3] attempted to solve the challenges of evaluating these complex systems as-a-whole, by integrating multiple simulation platforms with varying semantics and integrating and managing different simulation models and their interactions. Recently, a great interest has developed to use Functional Mockup Interface (FMI) [4] for a variety of dynamics simulation packages, particularly in the automotive industry. Leveraging the C2WT effort on effective integration of different simulation engines with different Models of Computation (MoCs), we propose, in this paper, to use the proven methods of High-Level Architecture (HLA)-based model and system integration. We identify the challenges of integrating Functional Mockup Unit for Co-Simulation (FMU-CS) in general and via HLA [5] and present a novel model-based approach to rapidly synthesize an effective integration. The approach presented provides a unique opportunity to integrate readily available FMU-CS components with various specialized simulation packages to rapidly synthesize HLA-based integrated simulations for the overall composed Cyber-Physical Systems.

*Keywords: Functional Mockup Interface, Functional Mock-up Unit for Co-Simulation, Cyber-Physical Systems, Heterogeneous simulation, Multi-paradigm*

*modeling, Model-based integration, DSML, Distributed Simulation, High-Level Architecture*

## 1 Introduction

Cyber-Physical Systems (CPS) [1] are composed of several collaborating physical and computing components that interact through embedded communication capabilities. These systems require advanced integration of abstractions and techniques that have been developed over the past years in disparate areas such as cyber systems that rely heavily on computation and networking and physical systems that employ various engineering methods in domains such as mechanical, thermal, electrical, electronic, hydraulic, thermal, biological, and acoustic.

Analysis of Cyber-Physical Systems poses unique challenges due to the heterogeneity of components and interactions [2]. The fundamental differences in the characteristics of these different physical and computation processes lead to a huge spectrum of modeling methods. For example, some components can be easily described by differential equations, while others like communication networks are better modeled as Discrete-Event Systems. As such, several simulation tools and techniques are needed for CPS simulation and analysis. This further necessitates an over-arching CPS model and system integration platform that is model-based and supports rapid synthesis of distributed heterogeneous CPS simulations.

Co-Simulation (Co-operative Simulation) is a simulation method that permits simulating individual components using different simulation tools simulta-

neously and collaboratively. Individual simulation tools exchange information such as system variables and their values, time steps for synchronization, and control signals for orchestrating the co-operative simulation. Thus, engineers can use different simulation tools together to create virtual prototypes of entire Cyber-Physical Systems. In practice, however, significant challenges remain with regard to the syntax and semantics of model and system integration.

In the Co-Simulation domain, a recent effort by the MODELISAR ITEA2 project that develops a tool independent standard called the Functional Mock-up Interface (FMI) [4] [7] [8] has gained significant influence, more prominently in the automotive industry. The FMI standard provides a well-defined specification and API to integrate simulation components. FMI-compliant simulations pack shared libraries that can be executed using the standardized function calls and the model execution must adhere to the rules of the standard. These function calls span all stages of the model execution, viz. initialization, configuration, access, modification, and manipulation.

The strength of FMI lies in the fact that all simulation tools participating in the Co-Simulation follow the defined standard and as such provides for standardized access to model equations. This permits coupling of Continuous-Time, Discrete-Time, and Discrete-Event that are part of a Cyber-Physical System. In some ways, this is also a limitation because not all simulation tools are amenable to support all of the strictly specified FMI function calls.

Another key requirement for Co-Simulation via FMI is to also develop a Master Algorithm (MA) that orchestrates the steps of Co-Simulation. Master algorithms have two tasks: (1) control the data exchange and (2) control time advancement among individual simulations according to the requirements of the integrated simulation of the overall Cyber-Physical System. Since the FMI standard does not describe or limit the implementation of the MAs, it basically leaves out the two fundamental challenges that all distributed simulation architectures face: *data exchange* and *time management*<sup>1</sup>. Solution for integrated data and time management in distributed simulations is technically complex and errors can easily lead to performance bottleneck and failures. This complexity pushes designers to adhere to the simplest solutions – losing much of the potential advantages of co-simulation.

---

<sup>1</sup> Theoretically, an MA for direct co-ordination among FMUs can be developed for each distributed simulation problem, but that would be expensive and subject to errors.

For example, Cyber-Physical Systems frequently involve vastly different sub-domains and physical processes that vary greatly in the execution frequency at which they need to run. This leads to significantly different dynamic response characteristics. For example, mechanical components of a complex CPS often have much slower frequency responses compared to fast electronic components. A single standalone monolithic model of a CPS therefore suffers heavily with solver inefficiencies. These systems are generally highly complex and have significant non-linearities and discontinuities, which further increases difficulties of gaining convergent solutions. Taking subsystems apart and using different solvers and step-sizes offers a potential solution. However, multirate composition also introduces inefficiencies due to time management, composition restrictions, data exchange, and potential stability issues if the system is split at the wrong place.

Challenges of distributed simulation have a long research history with notable results and standards. One of the widely used solutions developed by the U.S. Modeling and Simulation Coordination Office (M&S CO) is the High Level Architecture (HLA) [5]. The HLA provides a specification of a common technical architecture for modeling and simulation with a primary goal to facilitate interoperability among simulations and to promote re-use of simulations and their components. The HLA comprises of three major components: HLA rules, HLA interface specification, and HLA object model template [5]. With these rules, the HLA standardizes run-time support for various tasks, such as coordinated time evolution, message passing and shared object management. The key difference between FMI and HLA is that HLA provides a standard for data exchange and time management for individual simulation processes. This enables broader integration of different simulation tools using different Models of Computation. Functional Mock-up Units can be integrated as a participating simulation tool in the overall integrated simulation of the Cyber-Physical Systems.

A key benefit of HLA is that its Distributed Discrete Event model of computation allows full flexibility to individual subsystems in using any internal solver and model of computation. Moreover, this flexibility permits multirate simulations by design.

However, the HLA standard also lacks some key facilities for developing integrated distributed heterogeneous simulations. For example, the HLA standard does not formalize methods for developing interactions and objects used by HLA federates and it does not provide facilities for easily moving simulations from one computational node to other. Consequently, HLA-based simulations also require a sig-

nificant amount of tedious and error-prone hand-developed integration code.

Achieving the integrated simulation of Cyber-Physical Systems require effective integration of a huge spectrum of models of physical processes, communication systems, exchanged information, and control mechanisms. As detailed above, the approaches of FMI and HLA both have their advantages and some key limitations. The approach of using HLA as a master algorithm enables use of FMUs in a Co-Simulation environment while also providing flexibility of using other types of non-FMU simulations [9]. The resulting framework can provide a much broader scale of simulation tools that can be used in the integrated simulation of Cyber-Physical Systems. However, several gaps need to be filled in order to develop a platform that enables this integration in an efficient manner. A single efficient model-based platform is needed that:

- Enables modeling of interactions and shared objects between simulation tools
- Enables modeling and integration of systems with their data exchange mechanisms
- Enables modeling of deployment of simulation tools on distributed computational infrastructure
- Provides a standard master algorithm for FMI Co-Simulation
- Enables multirate modeling with dynamic time management
- Provides a set of tools to generate necessary artifacts for rapid synthesis of simulations

This paper attempts to address these important challenges in creating a single coherent platform for developing integrated distributed simulations of Cyber-Physical Systems. We build upon our previous work on a model-based integration platform called the Command and Control Wind Tunnel (C2WT) [2] [3].

The rest of the paper is organized as follows. Section 2 and 3 give an overview of the C2 Wind Tunnel and FMI for Co-Simulation respectively. We present our detailed model-based integration approach in Section 4 and provide a detailed case study with experimental results in Section 5. Section 6 provides the related work and Section 7 concludes the paper.

## 2 C2 Wind Tunnel

Over the past several years, we have developed a model-based multi-model integration platform called the Command and Control Wind Tunnel (C2WT) [2] [3]. It is an integrated, graphical, multi-model, dis-

tributed simulation environment for the experimental evaluation of large-scale C2 systems with various organizational and technical architectures. It enables a variety of simulation engines to interact and transmit data from one another and log and analyze simulation results. Figure 1 below gives a conceptual architecture of C2WT.

The High-Level Architecture is a standardized framework for distributed computer simulation systems. Communications between different federates is managed via the Run-Time Infrastructure (RTI) layer. The RTI provides a set of services such as time management, data distribution, message passing, and ownership management. Other components of the HLA standard are the Object Model Template (OMT) and the Federate Interface Specification (FIS).

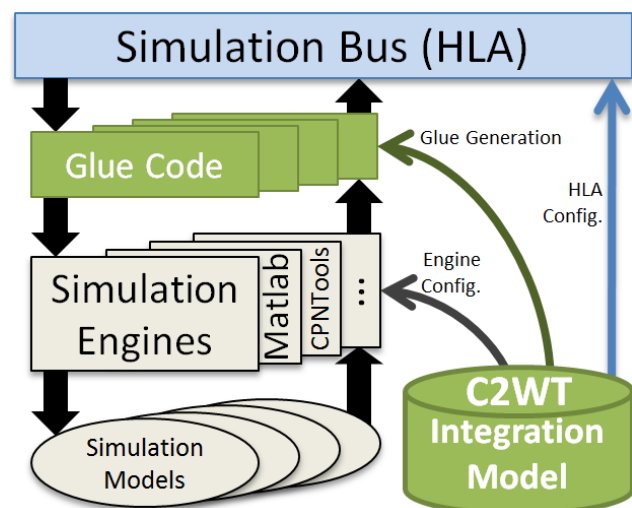


Figure 1: Conceptual architecture of C2WT

The HLA standard focuses on three primary areas. First is time coordination throughout the federation. The evolution of time is a key thread through each of the integrated simulators. Each simulation platform must slave its progression of time to that of the overall HLA clock. The HLA standard provides several methods by which to accomplish this. Second is coordination of inter-federate messages and shared data objects. The HLA standard provides a publish-and-subscribe mechanism for passing messages and object updates throughout the federation. Third, the HLA standard provides for basic simulation execution control. Starting, pausing, and stopping the execution of a simulation is built directly into the HLA standard. The C2 Wind Tunnel relies upon all of these services during run-time.

As HLA is an accepted standard, a number of commercial, academic, and alternate RTI implementations are available. Currently, we use the Portico

RTI [10] – which provides support for both C++ and Java clients and is compliant with version 1.3 of the HLA standard.

The HLA provides a standard for the RTI that supports the coordinated execution of distributed simulations. However, designing the model integration, coding the platform-to-RTI glue-code, and testing and deploying all of the various run-time components across multiple platform-specific simulation tools is a highly challenging task. C2WT provides a solution to this simulation integration problem. It provides a holistic modeling and management environment built around a custom Domain-Specific Modeling Language (DSML) [11], implemented in Generic Modeling Environment (GME) [11], and a related suite of model interpreters to coordinate between the integration model and the platform-specific simulation tools involved in the overall environment. It facilitates the rapid development of integration models and use of these models throughout the lifecycle of the simulated environment. With simulation engine specific model configurations and experiment specific deployment modeling, it enables significant automation in the development of integrated distributed simulation. With integration modeling support and various sophisticated generation tools, C2WT provides a robust platform for users to rapidly model and synthesize complex, heterogeneous, command and control simulations.

### 3 FMI for Co-Simulation

Functional Mock-up Interface (FMI) [4] [7] [8] was initiated and organized by Daimler AG within the ITEA2 project MODELISAR [4]. The FMI standard consists of two main parts. The first part is FMI for Model Exchange, which standardizes the distribution of a dynamic system model in the form of generated C-Code as an input/output block to other simulation environments. The second part is FMI for Co-Simulation, which standardizes the mechanisms for coupling of two or more simulation tools in a co-simulation environment.

The key idea is to have a discrete set of communication points only, at which times the subsystems exchange any data. Outside of these points, the subsystems are executed independently. The data exchange is controlled by a master system that also manages time synchronization of subsystems.

The FMI Co-simulation master simulator couples the subsystem simulators through a zip-archive. This zip-archive contains shared library files (.DLL, .SO) that conform to the function call specifications given in the standard. Each zip-archive also contains a

XML file that provides meta-data and further details of the model such as default start and stop times, variable types, units, tool specific data, parameter and variable names and attributes. The XML also contains specification for executing the model as a shared library during a simulation run (CoSimulation\_Standalone) or by importing a slave tool wrapper and interfacing it with the external tool (CoSimulation\_Tool).

## 4 Model-Based Integration

One of the primary contributions of our effort is our focus on developing a completely model-based integration approach. Our efforts leverage the Generic Modeling Environment (GME) [11] tool suite for designing the integration model DSML [11] and HLA [5] to provide run-time support as the “simulation bus”.

### 4.1 Needs and Challenges

Cyber-Physical Systems [1] [2] are highly complex and their simulation spans a multitude of computational domains and specializations. A large number of tools exist that have been developed for specific aspects of CPSs. A variety of tools exist even for a single aspect of CPSs. For example, many special purpose simulation tools exist to model and analyze vehicle dynamics or for switching mechanisms of hybrid drivetrains. As such the integration platform must be open toward use of any tool that may be required for some component/aspect of the CPS simulation.

A subtle problem in using multiple simulation tools in an integrated simulation is that they tend to use many different Models of Computation (MoC). For example, Discrete-Event, Discrete-Time, Continuous-Time, Synchronous Dataflow, are among the many MoCs used. Each MoC has a specific mechanism for time progression and event handling. The integration platform must be able to handle tools that use different MoCs in highly flexible manner. The integrated system must respect time synchronization with other simulation tools as well as the causality of events must be preserved. In addition to system integration, the platform must also enable integration of models by means of capturing the communication (with any translation that might be needed) that occurs between them.

In general, it is preferable to have a graphical environment that provides well-defined semantics for modeling concepts, their relations, and rules for composition. Moreover, for rapid synthesis of simu-

lations, the platform must support tools for translation of models to executable software that conform to specified executable semantics. The automation not only provides efficient development of simulations, it also significantly minimizes human errors.

The integration environment should also provide capabilities for modeling and configuration experimentation and logging.

Furthermore, when FMUs are integrated the rules of FMI must still be adhered to. Particularly, the models in the FMUs must be accessed, controlled, and manipulated using the function calls specified in the FMI standard.

## 4.2 Meta-modeling

The Generic Modeling Environment is a meta-programmable model-integrated computing (MIC) [11] toolkit that supports the creation of rich domain-specific modeling and program synthesis environments. Configuration is accomplished through meta models, expressed as UML class diagrams, specifying the modeling paradigm of the application domain. Meta models characterize the abstract syntax of the domain-specific modeling language, defining which objects (i.e. boxes, connections, and attributes) are permissible in the language. Another way to envision this is that a DSML [11] is a schema or data model for all the possible models that can be expressed by a language. Using finite state machines as an example, the DSML would consist of states and transitions. From these elements any state machine can be realized. The inherent flexibility and extensibility of the GME [11] via meta models make it an ideal foundation for the C2 Wind Tunnel environment. Alternate meta modeling frameworks have also been developed in the past, such as AToM3 [12], MetaCase [13], Microsoft DSL [14], and the Eclipse Modeling Framework [15].

## 4.3 Model-Based Integration of FMUs in C2WT

As detailed in section 2, C2WT provides an overarching modeling and management environment and a suite of model interpreters to coordinate the integration models and platform-specific simulation tools involved in the overall heterogeneous distributed simulations. The user is referred to [2] for details of the meta-modeling language and its executable semantics. In this section, we further discuss the integration of FMUs as HLA-federates in the C2WT platform.

In this work, the C2WT metamodel was further customized to enable FMU specific federate specifi-

cations. Although the original C2WT metamodel is sufficient to support integration of newer types of federates, having simulation tool/technique specific first-class objects in the modeling language makes reasoning about such entities more flexible and can support extensive automation. The FMU-federate model specifies the location of the zip archive, whether to log variable values during simulation, additional variables (other than input and output) to log, and ratio of macro and micro steps for multirate simulations.

Figure 2 below shows the extension to the original C2WT architecture to incorporate FMU federates in the platform.

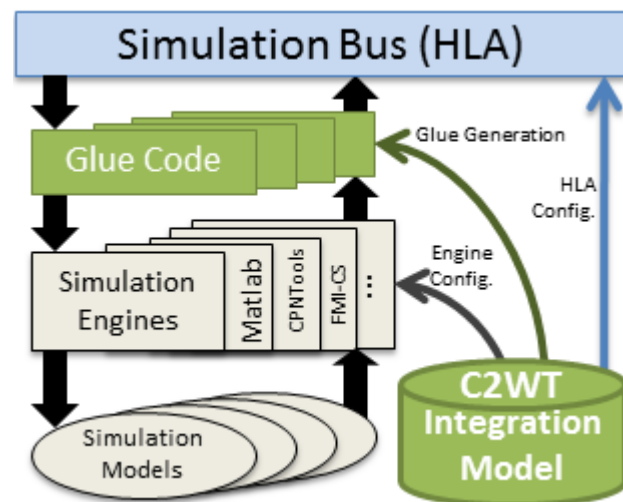


Figure 2: C2WT extended for FMI-CS

Our model interpreters can read the models with specified input and output relationships with other simulation tools and even other FMUs and can automatically generate all the executable code that can be deployed on different nodes in the available computational infrastructure for the simulation. As previously mentioned, C2WT supports simple modeling of computational infrastructure and assignment of federates on its nodes.

Following the rules of FMU access, modification, and manipulation as described in the FMI standard [4] [7], we developed a simplified procedure for FMU-federate execution as given below:

### Initialization phase (before simulation start):

- 1: Load FMU zip archive, read model description
- 2: Load shared libraries in the FMU
- 3: Instantiate the FMU slave
- 4: Setup input/output and HLA-interaction maps
- 5: Setup up logging

**Execution phase (during simulation):**

- 1: Synchronize start of simulation with all tools
- 2: Request RTI to proceed to step-size and wait
- 3: Update input variables with HLA updates
- 4: Call *doStep* in step-size/#micro-steps chunks
- 5: Continue #4 until full step-size is executed<sup>2</sup>.
- 6: Update HLA with output variables
- 7: Go to #2

Please note that above is rather simplified procedure of FMU integration mechanism in C2WT. The actual implementation also involves setting up statistical and database logging, micro-step management to avoid overlaps, error-handling, efficient federate code execution, reliable & reusable time advancing facilities, and model state and HLA interaction synchronization.

**5 Case Study**

To illustrate our model-based approach for FMU integration in C2WT we present a high-fidelity model of a representation of a Vehicle Thermal Management (VTM) system which is intended for studying interactions of thermal management systems within a vehicle.

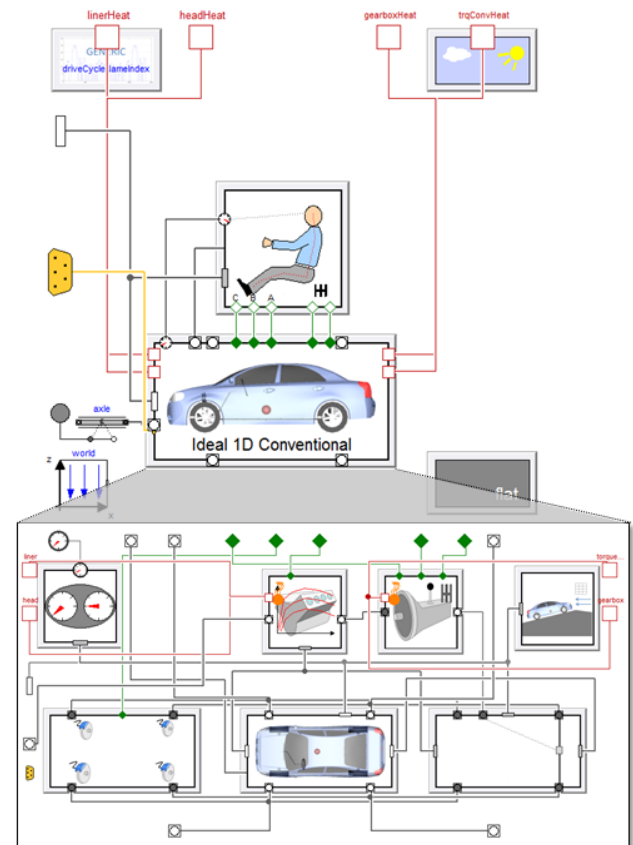
**5.1 Model description**

This particular example is a conventional four wheel chassis and drivetrain architecture with a spark ignition engine and standard transmission. These mechanical systems are created using components from the Vehicle Dynamics Library (VDL) from Modelon [16]. The model also includes a representation of the coolant loop for the engine and transmission oil loop in conjunction with a four heat exchanger stack for the thermal domain. These portions of the model are constructed from components of the Liquid Cooling Library (LCL) from Modelon. A snapshot of the overall model is shown in the following Figure 3 below.

The key component models of the system are: Driver, Vehicle (Engine, Transmission, Driveline, Chassis, Aerodynamics, External loads, and Brakes), Lumped engine thermal mass, Lumped transmission thermal mass, Engine coolant fluid circuit, Transmis-

sion oil cooling circuit, Heat exchanger stack, Low voltage battery, Alternator, Cooling fan and controller, and Grill shutters and controller. Table 1 below provides key features of these component models.

Since the purpose of this model is to study vehicle thermal dynamics, a simplified 1D longitudinal dynamics chassis model is used rather than a full 3D body model. This allows for faster simulations of the typically long duration drive cycles.



*Figure 3: Overall system model*

Component model	Key features
Driver	<ul style="list-style-type: none"> <li>• Closed loop speed control for drive cycle following with auto gear shifting</li> <li>• Standard list of drive cycles</li> </ul>
Vehicle - Engine	<ul style="list-style-type: none"> <li>• Torque map from throttle and speed with heat generation</li> </ul>
Vehicle - Transmission	<ul style="list-style-type: none"> <li>• Standard transmission with efficiency losses for heat generation</li> </ul>
Vehicle - Driveline	<ul style="list-style-type: none"> <li>• Rear wheel drive with parameterized final drive ratio</li> </ul>
Vehicle - Chassis	<ul style="list-style-type: none"> <li>• VDL compatible interfaces</li> <li>• 1D longitudinal dynamics</li> <li>• Ideal suspension and wheels</li> </ul>
Fluid coolant circuits	<ul style="list-style-type: none"> <li>• Heat absorption from thermal masses</li> <li>• Crankshaft pump loads</li> <li>• Thermostat fluid control</li> <li>• Fluid flow resistances</li> </ul>
Heat exchanger stack	<ul style="list-style-type: none"> <li>• Parameterized geometry, efficiency, resistances</li> <li>• Stack ordering effects</li> <li>• Air flow effects due to vehicle speed, grill position and fan speed</li> </ul>
Alternator	<ul style="list-style-type: none"> <li>• Crankshaft load</li> </ul>
Cooling fan and controller	<ul style="list-style-type: none"> <li>• Power supplied by alternator</li> </ul>
Grill shutters and controller	<ul style="list-style-type: none"> <li>• Variable position</li> <li>• Heat exchanger stack air flow modification</li> <li>• Aerodynamic drag effects</li> </ul>

*Table 1: Key features of component models*

<sup>2</sup> Note that an FMU freely updates its internal variables during micro-steps, but the input variables at the beginning of step #4, remain unchanged during step #5. Each subsequent micro-step, within the step #5, uses the updated internal variables in previous micro-step. Also, it is assumed that FMUs do not reject time-steps.



During the simulation, heat that is generated by the engine is stored within the engine thermal mass and then rejected to the coolant-to-air heat exchanger (radiator) through a coolant fluid loop. A similar loop and heat exchanger also exists for the transmission.

The model is well suited to thermal management controller design, studying tradeoffs between thermal management energy demands and fuel economy, heat exchanger efficiency and sizing, and coolant fluid flow dynamics.

For this paper, the model was partitioned into separate executables by dividing the model along domain boundaries. In this case the vehicle mechanics, electrical, and driver were grouped into one model while the fluid and thermal portions of the model were grouped into another. This partitioning allows for execution of Driver vehicle and Thermal management parts at different rates. Owing to the inclusion of fluid portions in the Thermal management part, this part needed to run with a much lower step-size than the Driver vehicle part to maintain system stability.

In order to do this the physical connections that are bisected by the boundaries must be converted to causal signals. As an example for the engine, the heat is generated within the mechanical portion of the model. The heat is directed to the lumped thermal model, within the thermal portion of the model, which determines the thermal mass temperature. Images of these two systems are shown in Figures 4 and 5 below.

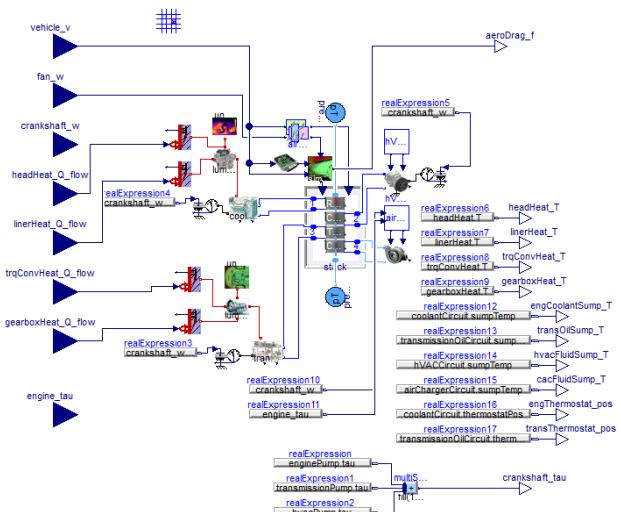


Figure 5: Thermal management model

### 5.2 Simulation architecture

The simulation setup consisted of mainly three federates, viz. Driver vehicle, Thermal management, and the Manager federate. Manager federate is an auto-generated external federate, which is used mainly as a front-end controller of the overall heterogeneous simulation. The simulation architecture is illustrated in the Figure 6 below.

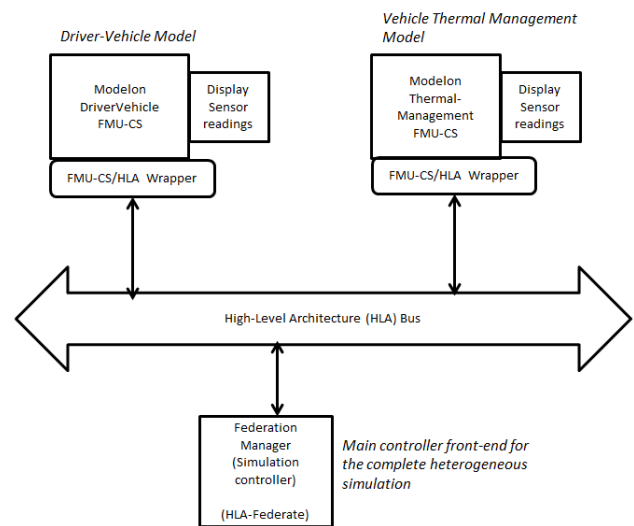


Figure 6: Simulation architecture

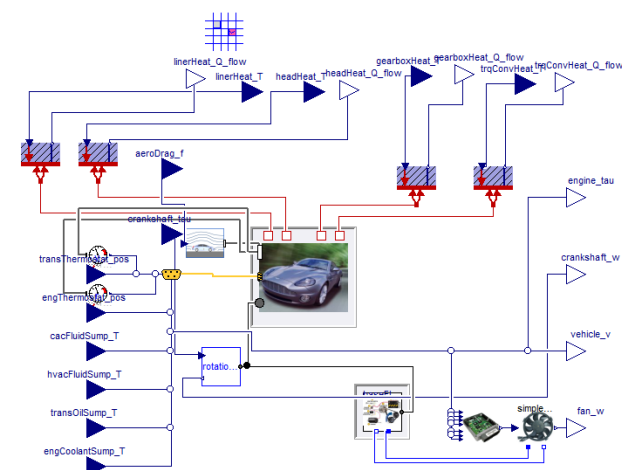


Figure 4: Driver vehicle model

### 5.3 Data and Integration model

The actual data and integration model are given in the Figures 7 and 8 below. These show the input and variables from the Driver vehicle and Thermal management federates. These two models are executed as FMUs in the C2WT.

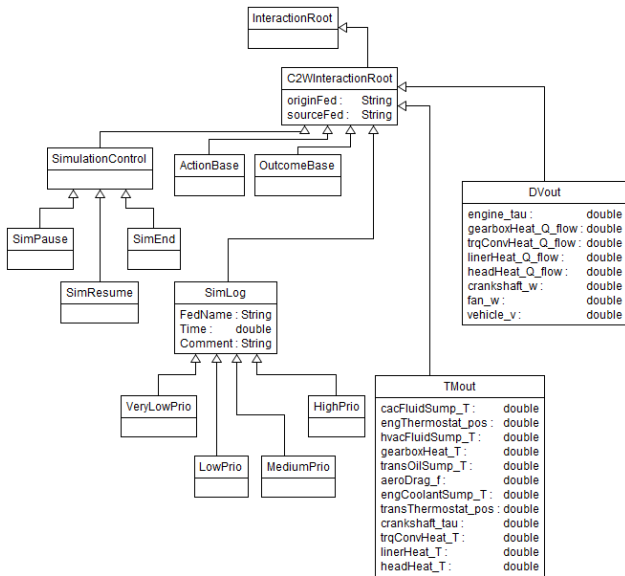


Figure 7: Data model

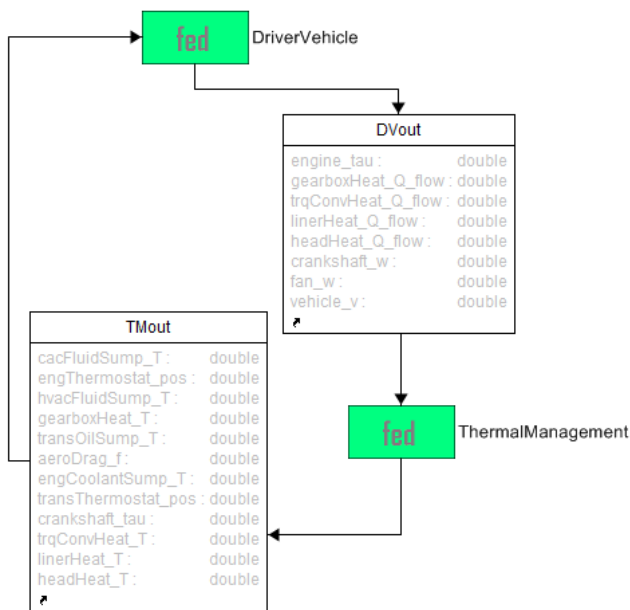


Figure 8: Integration model

### 5.4 Experimental Results

For the experiment, the Driver vehicle and Thermal management FMUs were exported from Dymola [16] models by Modelon, Inc. [16]. We used a JFMI Ptolemy APIs [17] to connect the FMUs to our Java based C2WT platform. All federates were running in a single Ubuntu 32 virtual machine. The Run-Time Infrastructure (RTI) used was Portico [10]. Total simulation time for the experiment was 50 seconds.

The simulation was setup as a multirate simulation with different step-sizes for the three federates: Driver vehicle (10 ms), Thermal management (5 ms), and Federation Manager (100 ms). The entire simulation ran in about ~9 minutes. The Figures 9 and 10 above show the experimental results for the total 50 seconds of simulation time. It should be noted though that the VTM models used were currently not optimized for efficiency.

From the experimental results, we found closely matching plots with same peak and trough values that were in the equivalent single monolithic (combined Driver vehicle and Thermal management) model. The overall runtime (~9 minutes) was also comparable to standalone single model simulation time in Dymola (~6 minutes) despite the use of a third federate (viz. Manager federate) in the simulation and delays due to inter-process communications.

The models were developed with a variable step solver as requirement. However, they could still run with a fixed step solver (with a maximum step-size of 1.5 ms). However, with our setup of separating the Driver vehicle and Thermal management components as separate FMUs and executing them through C2WT platform, we could even execute these components at 10 ms and 5 ms step-sizes respectively.

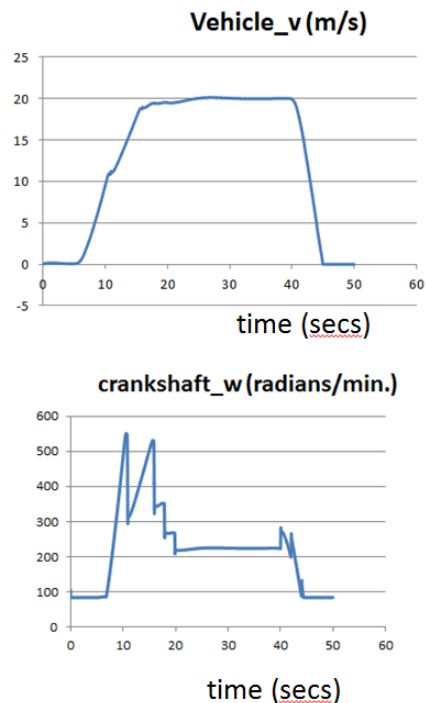


Figure 9: Vehicle speed and crankshaft angular velocity

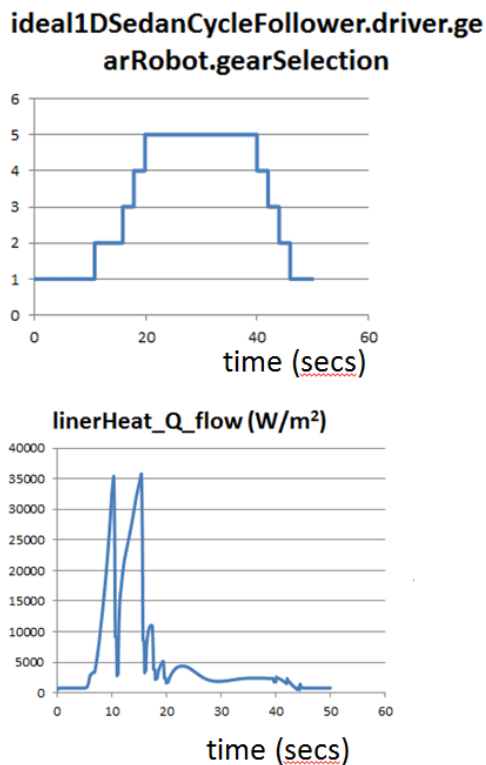


Figure 10: Gear selection and Liner heat flux

Yet another experiment we have performed is the one where we placed a network simulator for the CAN bus that must be placed between the above two components. We used the OMNeT++ simulator [18] to model that. In this experiment, we varied the rates of the FMUs to initially match the rate at which network simulator was run, viz. 0.5 ms, and then in the second setup we increased the step-size of Driver vehicle and Thermal management to 1 ms. We found that the results still matched while in the second setup they executed in about one-third the overall wall-clock time. We omit further details of experiment setup here for brevity. In the future, we plan to further publish our work focusing specifically on the multirate aspects in distributed CPS simulations.

## 6 Related Work

Distributed simulation of Cyber-Physical Systems [1] is a highly challenging endeavor with its unique set of requirements. In the past a number of researchers have attempted to solve these problems.

Some approaches aim at providing an integration mechanism for two or three tools used in CPS simulation. For example, [20] shows integrated CPS simulation by coupling Matlab and a network simulation called EPANET, [21] presents an integration approach using SysML and numerical solvers, and [22]

presents a co-simulation platform using Modelica framework and the ns-2 network simulator. There are also approaches to provide a complete dedicated simulation environment. For example, [23] [24] presented a unified software environment for model-based integration and multi-granular evaluation of widely used simulators. A simple master algorithm for Co-Simulation using FMI was given in [25] and a framework providing a central master simulator for FMUs as well as live components was provided in [26]. A dedicated environment for modeling with different models of computations and their execution via HLA was presented in [27]. Also, a co-simulation tool-chain has been proposed in [28], where existing system components from domains such as TinyOS and Modelica can be imported into SysML and then converted into corresponding FMU blocks. It was recently argued in [29] that it may be feasible to create a HLA-based master algorithm for FMI Co-simulations and then later a method was proposed in [9] [30] [31].

Our work focusses on providing a framework to enable model-based rapid synthesis on CPS simulations. The origins of our model-based approach can be found in [32]. Further, [33] explains how model-based rapid synthesis can lead to development of Cyber-Physical Systems with continuous integration techniques. The C2WT framework [2] developed at our institute facilitates model-based rapid integration of heterogeneous and distributed simulations. Recently, C2WT framework was used to develop an evaluation tool for networked control systems in [34].

## 7 Conclusions

In this paper, we have successfully demonstrated a model-based integration approach to rapidly synthesize multi-model distributed simulation that may also involve co-simulation FMUs as component models. The FMUs are automatically wrapped as HLA-federates that can be executed in the C2WT platform.

We also illustrated that different federates can be run with different clocks and their synchronization in C2WT is managed using HLA time management facilities. We have also integrated FMU-CS in simulations that also use other simulation tools such as a network simulator or a 3D terrain simulator. The integration of other federates in C2WT has been previously demonstrated in [2]. Thus C2WT provides a broader range of simulation tool integration that involves FMI and non-FMI simulations to enable development of System-of-System (SOS) simulations.

C2WT supports real-time and as-fast-as-possible modes of simulation execution. However, currently the real-time simulation requires that the individual component simulations can run faster than real-time.

C2WT also supports human-in-the-loop simulations with real-time simulations. In this case human interaction with running simulations (e.g. in military training exercises) is performed using HLA-interaction mappings.

One of the key benefits of C2WT platform is its support for extensive experimentation, message logging, state variables logging, and analysis support.

The research at our institute is currently ongoing with the applications of FMI Co-Simulation using HLA-based integrations. We anticipate novel methods for FMI Co-Simulations that are rapidly synthesized and may perform faster than single monolithic simulations.

We are also working on extending the C2WT platform to support other simulation techniques and tools such as SystemC.

## 8 Acknowledgements

The authors acknowledge financial support from the US DoD's Defense Advanced Research Projects Agency under the project "Adaptive Vehicle Make" [19]. We also acknowledge the invaluable contributions for the efforts in this paper from our collaborators at Modelon, Inc. [16].

## References

- [1] J. Sztipanovits, "Composition of cyber-physical systems," 14th Annual IEEE Int'l. Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07). Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–6.
- [2] Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, Gabor Karsai, "Rapid Synthesis of High-Level Architecture-Based Heterogeneous Simulation: A Model-Based Integration Approach", *Simulation* 88(2), 217-232 (2012)
- [3] C2WT community wiki – [wiki.isis.vanderbilt.edu/OpenC2WT](http://wiki.isis.vanderbilt.edu/OpenC2WT)
- [4] Functional Mock-up Interface – [www.fmi-standard.org](http://www.fmi-standard.org)
- [5] HLA standard – IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) – framework and rules [ieeexplore.ieee.org/servlet/opac?punumber=7179](http://ieeexplore.ieee.org/servlet/opac?punumber=7179).
- [6] E. Lee, "Cyber physical systems: Design challenges," in Proc. of the 11th IEEE Int'l. Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08), May 2008, pp. 363–369.
- [7] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. V. Peetz, S. Wolf, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models", in 8th International Modelica Conference, Dresden, 2011, pp. 20-22.
- [8] Modelica Association: Modelica – A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2, March 24, 2010: [www.modelica.org/documentas/ModelicaSpec32.pdf](http://www.modelica.org/documentas/ModelicaSpec32.pdf)
- [9] Awais, M.U.; Palensky, P.; Elsheikh, A.; Widl, E.; Matthias, S., "The high level architecture RTI as a master to the functional mock-up interface components," *Computing, Networking and Communications (ICNC), 2013 International Conference on*, vol., no., pp.315,320, 28-31 Jan. 2013 doi: 10.1109/ICCNC.2013.6504102
- [10] Portico RTI - [www.porticoproject.org](http://www.porticoproject.org)
- [11] Sztipanovits, J., and Karsai, G. 1997. "Model-Integrated Computing", *IEEE Computer*, 30(110-112)
- [12] de Laura, J., and Vangheluwe, H., 2002. "AToM3: A Tool for Multi-formalism and Meta-Modeling", *Lecture Notes in Computer Science*, 2306 (174-188).
- [13] Tolvanen, J.P., and Lyytinen, K. 1993. "Flexible Method Adaptation in CASE. The Metamodeling Approach", *Scandinavian Journal of Information Science*, v5 n1 (71-77).
- [14] Cook, S., Jones, G., Kent, S., and Wills, A. 2007. "Domain-specific Development with Visual Studio DSL Tools", Addison-Wesley Professional
- [15] The Eclipse Foundation – [www.eclipse.org](http://www.eclipse.org)
- [16] Modelon, Inc. – [www.modelon.com](http://www.modelon.com)
- [17] JFMI: A Java wrapper for the Functional Mockup Interface – [www.ptolemy.eecs.berkeley.edu/java/jfmi](http://www.ptolemy.eecs.berkeley.edu/java/jfmi)
- [18] OMNeT++ - [www.omnetpp.org](http://www.omnetpp.org)

- [19] DARPA Adaptive Vehicle Make Program – [www.darpa.mil/Our\\_Work/TTO/Programs/Adaptive\\_Vehicle\\_Make\\_\\_\(AVM\).aspx](http://www.darpa.mil/Our_Work/TTO/Programs/Adaptive_Vehicle_Make__(AVM).aspx)
- [20] Jing Lin; Sedigh, S.; Miller, A., “Towards Integrated Simulation of Cyber-Physical Systems: A Case Study on Intelligent Water Distribution,” Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on, vol., no., pp.690,695, 12-14 Dec. 2009. doi: 10.1109/DASC.2009.140.
- [21] Palachi, E.; Cohen, C.; Takashi, S., “Simulation of cyber physical models using SysML and numerical solvers,” Systems Conference (SysCon), 2013 IEEE International, vol., no., pp.671,675, 15-18 April 2013. doi: 10.1109/SysCon.2013.6549954.
- [22] A. Al-Hammouri, V. Liberatore, H. Al-Omari, Z. Al-Qudah, M. S. Branicky, and D. Agrawal, “A co-simulation platform for actuator networks,” in Proc. of the 5th Int'l. Conference on Embedded Networked Sensor Systems (SenSys '07). New York, NY, USA: ACM, 2007, pp. 383–384.
- [23] A. Bakshi, V. K. Prasanna, and A. Ledeczki. 2001, “MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems,” SIGPLAN Not. 36, 8 (August 2001), 82-93. DOI=10.1145/384196.384210.
- [24] Agrawal, A.; Ledeczki, A., “Multigranular simulation of heterogeneous embedded systems,” Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the , vol., no., pp.3,10, 7-10 April 2003. doi: 10.1109/ECBS.2003.1194776.
- [25] J. Bastian, Ch. Clauß, S. Wolf, P. Schneider, “Master for Co-Simulation Using FMI,” 8th International Modelica Conference. Dresden 2011.
- [26] De Filippo, F.; Stork, A.; Schmedt, H.; Bruno, F., “A modular architecture for a driving simulator based on the FDMU approach,” International Journal on Interactive Design and Manufacturing (IJIDeM) (2013): 1-12 , March 09, 2013.
- [27] Lasnier, Gilles and Cardoso, Janette and Siron, Pierre and Pagetti, Claire and Derler, Patricia, “Distributed Simulation of Heterogeneous and Real-time Systems,” In 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications - IEEE/ACM DS-RT 2013, 30 October 2013 - 01 November 2013 (Delft, Netherlands).
- [28] B. Wang and J. S. Baras, “HybridSim: A Modeling and Co-simulation Toolchain for Cyber-Physical Systems,” Proceedings of 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp. no. 33-40, Delft, Netherlands, October 30 - November 1, 2013.
- [29] Elsheikh, A.; Awais, M.U.; Widl, E., Palensky, P., “Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface,” Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on , vol., no., pp.1,6, 20-20 May 2013. doi: 10.1109/MSCPES.2013.6623315.
- [30] Awais, Muhammad Usman; Palensky, Peter; Mueller, Wolfgang; Widl, Edmund; Elsheikh, Atiyah, “Distributed hybrid simulation using the HLA and the Functional Mock-up Interface,” Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE , vol., no., pp.7564,7569, 10-13 Nov. 2013. doi: 10.1109/IECON.2013.6700393.
- [31] Awais, Muhammad Usman, Wolfgang Mueller, Atiyah Elsheikh, Peter Palensky, and Edmund Widl, “Using the HLA for Distributed Continuous Simulations,” In proceeding of: The 8th EUROSIM Congress on Modelling and Simulation. Sep. 2013.
- [32] Karsai, G.; Sztipanovits, J.; Ledeczki, A.; Bapty, T., “Model-integrated development of embedded software,” Proceedings of the IEEE , vol.91, no.1, pp.145,164, Jan 2003. doi: 10.1109/JPROC.2002.805824.
- [33] Gabor Karsai , Janos Sztipanovits, “Model-Integrated Development of Cyber-Physical Systems,” Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems, October 01-03, 2008, Anacardi, Capri Island, Italy doi: 10.1007/978-3-540-87785-1\_5.
- [34] D. Riley, E. Eyisi, J. Bai, X. Koutsoukos, Y. Xue, and J. Sztipanovits, “Networked control system wind tunnel (NCSWT): an evaluation tool for networked multi-agent systems,” in Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools), 2011, pp. 9–18.



# Adapting Functional Mockup Units for HLA-compliant Distributed Simulation

Faruk Yılmaz, Umut Durak, Koray Taylan

Halit Oğuztüzün

Roketsan Missiles Inc.  
Ankara, Turkey

[fyilmaz|udurak|ktaylan]@roketan.com.tr

Middle East Technical University  
Ankara, Turkey

oguztuzn@metu.edu.tr

## Abstract

Conceptual design of systems requires aggregate level simulations of the designed system in its operational setting. By this way, performance of the system and its interactions with the other entities in its environment can be evaluated. The complex nature of these simulations often requires distributed execution. IEEE 1516 High Level Architecture (HLA) is a widely accepted standard architecture for distributed aggregate level simulations. Functional Mock-up Interface (FMI) is a recent standardization effort that leads to a tool independent systems simulation interface that enables model reuse and co-simulation. This paper aims to present a method for developing HLA-compliant federates using FMI. The method enables a Functional Mock-up Unit to join an HLA-compliant federation as a member.

*Keywords: Functional Mockup Interface; High Level Architecture; Distributed Simulation*

## 1 Introduction

Systems development process starts with conceptual design phase in which designers create concepts and conduct trade off analysis. Modeling and simulation have always been essential tools for conceptual design. Early stage systems modeling aims to identify the system requirements and its interactions with its operating environment. Effect based models, integrated in a large scale operational settings are used to evaluate the performance of the system concerning the accomplishment of its mission. Simulation of the mission space of a system requires modeling large

number of entities and often simulating them in a distributed fashion. IEEE 1516 High Level Architecture (HLA) standard [1] [2] [3] is commonly used to integrate loosely coupled models of the entities in a mission space.

The Functional Mock-up Interface (FMI) is a newly developed, tool-independent model interface standard [4] [5]. Its main purpose is to model reuse between various modeling tools and environments throughout the systems development phases. A simulation component conforming to FMI is called a Functional Mock-up Unit (FMU), whose contents include a model description file, user defined libraries, source codes, model icons and documentation.

FMI and HLA has completely different behavior. While HLA supports to work at process level, the master of the FMU does not care about the topics such as entity transfer, shared resource management, time synchronization or ownership management [10].

On the other hand, there is a potential to reuse existing FMUs as federates in an HLA-compliant distributed simulation, i.e. federation. By this way, FMI will also serve as a model interface for distributed simulation entities in the concept of design phase. Here in this study, we introduce a mechanism to develop Functional Mockup Unit Federates (FMUFD) from FMUs.

### 1.1 Related Work

As model based development of engineering systems are getting more popular, connecting engineering models to the distributed simulation environments is also becoming an important issue of concern [6][7]. There have been some attempts for developing such tools and methodologies. Closely related to our work,

there exist two particular efforts providing a mechanism for connecting models to HLA environments.

MatlabHLA-Toolbox [8] and HLA Blockset [9] are available toolboxes to provide HLA communication feature to the Matlab. With these toolboxes, modelers can create a federate, join a federation and start publishing and subscribing entities and events. However, these solutions can only work in Matlab environment.

In [10], authors introduce an approach to run FMI Co-Simulation environment over HLA. They employ HLA RTI as a master to synchronize the simulation that is composed entirely of FMUs. They define an Object Class derived from the interface specifications of all participating federates and let each federate out of an FMU publish or subscribe its required attributes. In contrast, our approach enables participation of FMUs in a federation with non-FMU members as well.

## 2 Functional Mockup Interface

Functional Mockup Interface provides an interface specification for simulation components called Functional Mockup Units. FMI provides two standard interfaces, namely, FMI for Co-Simulation and FMI for Model Exchange [4] [5].

While FMI for Model Exchange specifies the interface for callers with explicit or implicit integrators, FMI for Co-Simulation specifies the interface for simulation runnables that possess solvers in them. As we can view HLA Federates as standalone simulation runnables, this effort is based on FMI-Co-Simulation interface for federate development. In this work, the first version of the standard is used as the baseline [5].

### 2.1 FMI for Co-simulation

As mentioned above, FMI for Co-Simulation is a standard interface for the model output containing its solver inside. Therefore, the user does not need to know which integration method is actually employed to solve the ordinary differential equations within the model.

For each of the FMU in a co-simulation environment, the communication capabilities are configured in a model specific XML file, namely *ModelDescription.xml* file. Communication with an FMU can only be realized in a discrete communication point, which is a sampling point or a synchronization point of the FMU [5].

#### 2.1.1 Computational Flow

As show in Figure 1, FMU co-simulation computational flow has three main states, namely Instantiation and Initialization, Running and Termination.

##### Instantiation and Initialization

A new FMU instance is created and initiated to be ready to run. Memory allocations and initial value setting for the FMU parameters are performed in this phase.

##### Running

In this phase, FMU model is executed via calling *doStep()* method. Intuitively, before running a step, FMU input parameters are set by calling *FMUSetXXX(...)* and after the completion of this step the model output parameter are read by the master via calling *fmiGetXXX(...)*.

##### Termination

The model component is unloaded and the memory is cleaned up in this phase.

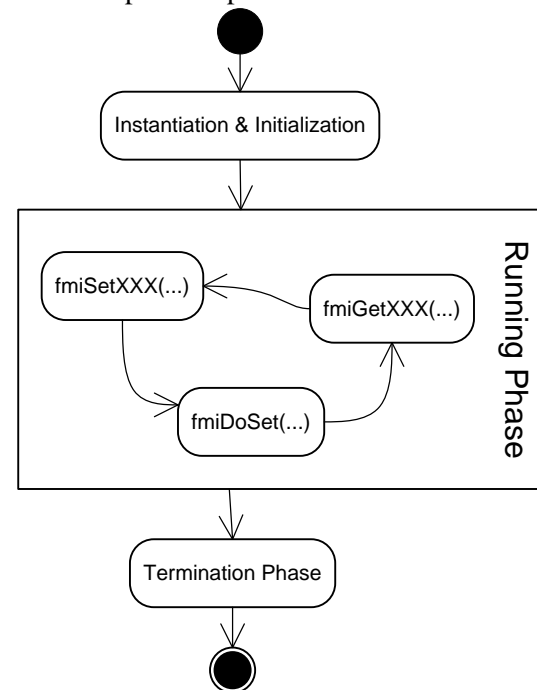


Figure 1 – FMU Co-Simulation Model Computational Flow

## 3 High Level Architecture

The High Level Architecture (HLA) is a common framework for distributed simulation systems. HLA promotes interoperability between heterogeneous simulations and supports the reuse of models in different contexts. HLA provides communicating data and synchronization actions between simulation members regardless of their computing platforms [1].

HLA combines simulations (federates) into a larger simulation (federation), where federates are



components and federations are component based applications. The HLA requires runtime infrastructure (RTI) software to support the operation of a federation execution. RTI provides a set of services and by using these services a federate can interact with the federation at runtime. How a federate can utilize these services is defined by the Federate Interface Specification [2].

Federation Object Model (FOM) is the HLA Federation Object Model that describes all of the object classes and interactions, attributes of object classes and parameters of interactions for the federation. Also, FOM establishes the information model contract which governs the simulation. Simulation Object Model (SOM), on the other hand, is the HLA Simulation Object Model that describes the object classes and interactions, attributes of object classes and parameters of interactions information which are exposed or consumed by a federate [2].

### 3.1 HLA Services

HLA provides six groups of services to enable distributed simulation in an aggregate level [2]. Federation Management describes how to create, join, resign and manage federations, save and restore federation states. Declaration Management defines the publishing and subscribing to objects and attributes. Object Management service states how to register new instance of an object class or interaction, update the attributes, receive interactions, discover new instances and receive updates of attributes. Ownership Management defines acquisition of ownership of the registered objects. Time management describes how a federate can advance its logical time along with other federates and how to deliver the time-stamped events ensuring that a federate can never receive an event with a timestamp less than the federate's logical time. Data distribution management defines the production and consumption of data to bind the relevance of communication data among federates. As a result, RTI can recognize the irrelevant data and prevents its delivery to consumers.

### 3.2 HLA Object Model

HLA provides object classes and interactions as the object models, which are used to publish/subscribe the data over distributed simulation environment. Providing the data exchange between federates is one of the responsibilities of the RTI.

An **object class** can be derived from another object class. *HLAObjectRoot* is the base class of the all object classes. Each object class can contain one or more at-

tributes. Derived classes also inherit base class attributes. Attributes have data types. A federate will publish/subscribe only interested attributes of an object class; it does not have to deal with all the attributes in an object class.

An **interaction** can be derived from another interaction. *HLAInteractionRoot* is the base class of the all interactions. Each interaction contains one or many object parameters. Derived interactions takes base interaction parameters also. Parameters have data types. A federate should fill all the parameters of an interaction to publish it.

HLA provides six different **data types** where user can create variety of data structures by using those data types. The published/subscribed values are stored in these data structures. The details of data types are given below [3]:

- **Basic Datatype:** Basic data refers to a predefined set of data representations. Following data types should be defined by any OMT: *HLAinteger16BE*, *HLAinteger32BE*, *HLAinteger64BE*, *HLAfloat32BE*, *HLAfloat64BE*, *HLAoctetPairBE*, *HLAinteger16LE*, *HLAinteger32LE*, *HLAinteger64LE*, *HLAfloat32LE*, *HLAfloat64LE*, *HLAoctetPairLE*, and *HLAoctet*.
- **Simple Datatype:** The simple data type table refers to simple, scalar data items. Following data types should be defined by any OMT: *HLAASCIIchar*, *HLAunicodeChar*, and *HLAbyte*.
- **Enumerated Datatype:** The enumerated data type refers to data elements that can take on a finite discrete set of possible values. Following data type should be defined by any OMT: *HLAboolean*.
- **Array Datatype:** The array data type table refers to indexed homogenous collections of data types; these constructs are also known as arrays or sequences. Following data types should be defined by any OMT: *HLAASCIIstring*, *HLAunicodeString*, and *HLAopaqueData*.
- **Fixed Record Datatype:** The fixed record data type table refers to heterogeneous collections of types; these constructs are also known as records or structures. This allows users to build structures of data according to the needs of their federate or federation.
- **Variant Record Datatype:** The variant record data type table refers to discriminated unions of types; these constructs are also known as variant or choice records.

### 3.3 HLA Padding Rules

HLA requires that certain types of data start at a particular kind of location. Therefore, usually there is a requirement for extra bytes, namely padding bytes,

between data fields in a structure. To illustrate, consider a structure where the first field is a byte and second field is a double. Double must start at a position which is a multiple of 8. Therefore, seven bytes of padding is needed between byte field and double field for a proper structure.

The padding rules are used to determine exact positions of the fields of a data type, which constructs the data structure of an attribute.

These rules for constructed data types (arrays, fixed records, and variant records) as described below [3]:

#### Base Datatype

Each base type has a boundary value as provided in table 1. During the calculation of padding, this table is used to calculate structure boundary value.

Table 1 – Basic Datatype Boundary Values

Basic representation	Octet Boundary Value
<i>HLAoctet</i>	1
<i>HLAoctetPairBE</i>	2
<i>HLAinteger16BE</i>	2
<i>HLAinteger32BE</i>	4
<i>HLAinteger64BE</i>	8
<i>HLAfloat32BE</i>	4
<i>HLAfloat64BE</i>	8
<i>HLAoctetPairLE</i>	2
<i>HLAinteger16LE</i>	2
<i>HLAinteger32LE</i>	4
<i>HLAinteger64LE</i>	8
<i>HLAfloat32LE</i>	4
<i>HLAfloat64LE</i>	8

#### Simple Datatype

Same base data type padding rules also apply for simple datatype.

#### Enumerated Datatype

Same base data type padding rules also apply for enumerated datatype.

#### Fixed Record Datatype

The padding bytes are added to each field when necessary to ensure that the next field in the record is properly aligned. After a field the padding bytes can be calculated by using the following formula:

$$(Offset_i + Size_i + P_i) \bmod V_{i+1} = 0$$

where  $Offset_i$  refers to the offset of the  $i$ 'th field of the record as bytes,

$Size_i$  refers to the size of the  $i$ 'th field of the record as bytes,

$V_{i+1}$  is the octet boundary value of field  $(i + 1)$ th of the record.

#### Variant Record Datatype

The *HLAvariantRecord* encoding shall consist of the discriminant followed by a field. This field is chosen by using the value of discriminant. The discriminant is placed at offset 0 of the record. The padding bytes  $P$  are calculated by using the following formula:

$$(Size + P) \bmod V = 0$$

where  $Size$  refers to the size of the discriminant as bytes, and  $V$  refers to the maximum of the octet boundary values of the alternatives.

#### HLA Array Datatype with Fixed Cardinality

The padding bytes  $P_i$  between  $i$ 'th and  $(i+1)$ th elements can be calculated by using following formula:

$$(Size_i + P_i) \bmod V = 0$$

where  $Size_i$  is the size of the  $i$ 'th element of the array in bytes,

$V$  is the octet boundary value of the element type.

#### HLA Array Datatype with Variant Cardinality

The first 4 bytes are used to present the number of the elements in the array. These 4 bytes are encoded as *HLAinteger32BE*. The padding bytes can be added between the inform element and the first element of the sequence. The padding bytes can be found by using following formula:

$$(4 + P) \bmod V = 0$$

where  $V$  is the octet boundary value of the element type.

## 4 Functional Mockup Unit Federate Design

The FMI for Co-Simulation standard does not provide a specification for connecting FMUs to an HLA federation, hence, FMI Co-Simulation does not have an interface ready to utilize HLA services. Moreover, there is no convenient way to convert FMU scalar variables to HLA object class attributes, because FMI Co-Simulation only supports the following primitive types: *real*, *integer*, *string*, *Boolean* and *Enumeration*. On the other hand, HLA attributes can represent any data type structure, from basic data types to the complex data type structures. Since FMI Co-Simulation scalar variables can only map to HLA basic data types, a simulation environment using complex data types cannot be directly supported by FMI Co-Simulation.

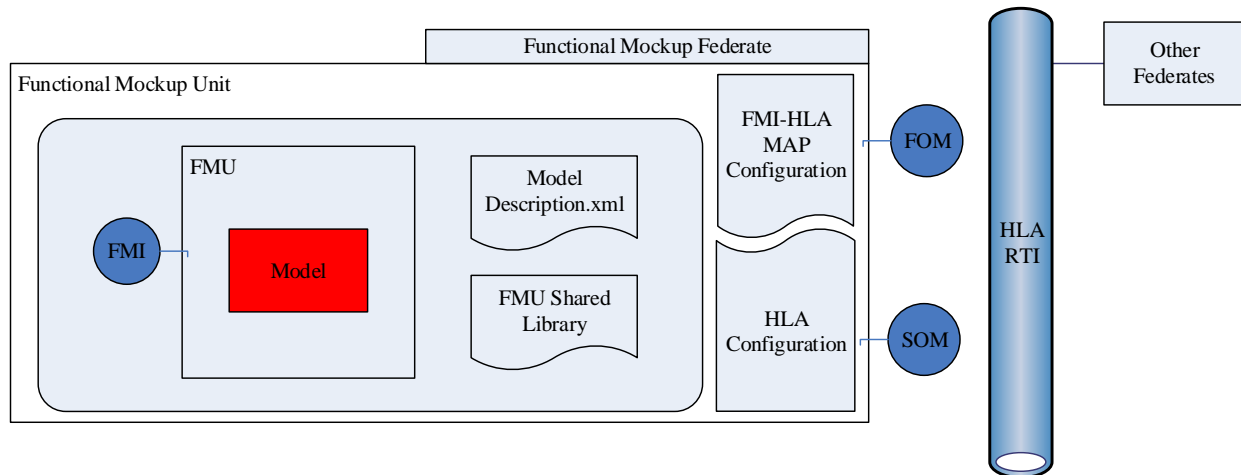


Figure 2 – Functional Mockup Unit Federate

Hence, there is a need for a wrapper that connects an FMU, in the context of FMI Co-Simulation, to the HLA distributed simulation environment. The work conducted handles the problem by designing a Functional Mockup Unit Federate (FMUFd). FMUFd has the following responsibilities:

- Instantiating, initializing, stepping and terminating of a FMU model.
- Providing the communication of distributed environment with services by using the HLA standard interface.
- Converting FMU model outputs to compatible HLA data types and sending them as HLA object updates.
- Receiving model inputs from HLA objects and converting them to compatible FMU types.

The top level structure of FMUFd that satisfies these requirements is depicted in Figure 2. The FMUFd is composed of the FMU model, FMI-HLA Map configurations and HLA connection configuration. FMI-HLA Map Configuration is used to inform FMUFd about HLA FMI relation. For each FMU, using this structure an FMUFd is needed to be configured. HLA connection configuration is related with the federation and FOM information of distributed simulation environment. By using these data, FMUFd runs with stepwise activities. As shown in Figure 3, these activities can be grouped into four main phases, namely, initialization, object discovery, stepping and termination.

In **initialization** phase, FMUFd loads and initializes the FMU and then connects the HLA federation as a federate with related HLA services and declare interested object classes for publishing and subscribing.

In **object reflection** phase, the subscribed object class instances are discovered and their values are reflected.

The **stepping** phase is the main phase of the simulation. In this phase, FMU input variables are reflected from related HLA objects, FMU runs one time step, and then, FMU output values are reflected to related HLA objects.

In **termination** phase, FMUFd terminates and unloads FMU, resigns from federation, frees allocated memory and finally stops.

The details of these steps with the process of connecting FMU to the HLA simulation environment will be described in the following sections. After that, the FMUFd capabilities in terms of the HLA services and FMUFd limitations will be mentioned briefly.

#### 4.1 Loading an FMU

The loading of FMU takes two phases: In the first phase, the model description file is parsed, while the FMU is loaded and initialized in the second phase.

FMU Model description file provides the static information of all exposed variables and model related data. FMUFd uses model description file to identify scalar variables with data types and value reference, Globally Unique Identifier (GUID) and the model name. The scalar variables are used in data flow between FMUFd and FMU model. GUID is used for validating concrete coded FMU with model description file. Model name is used to load shared object and FMI functions. The dynamic link library has the same name as the model; shared object FMI functions should also take the model name as a prefix to their functions [5].

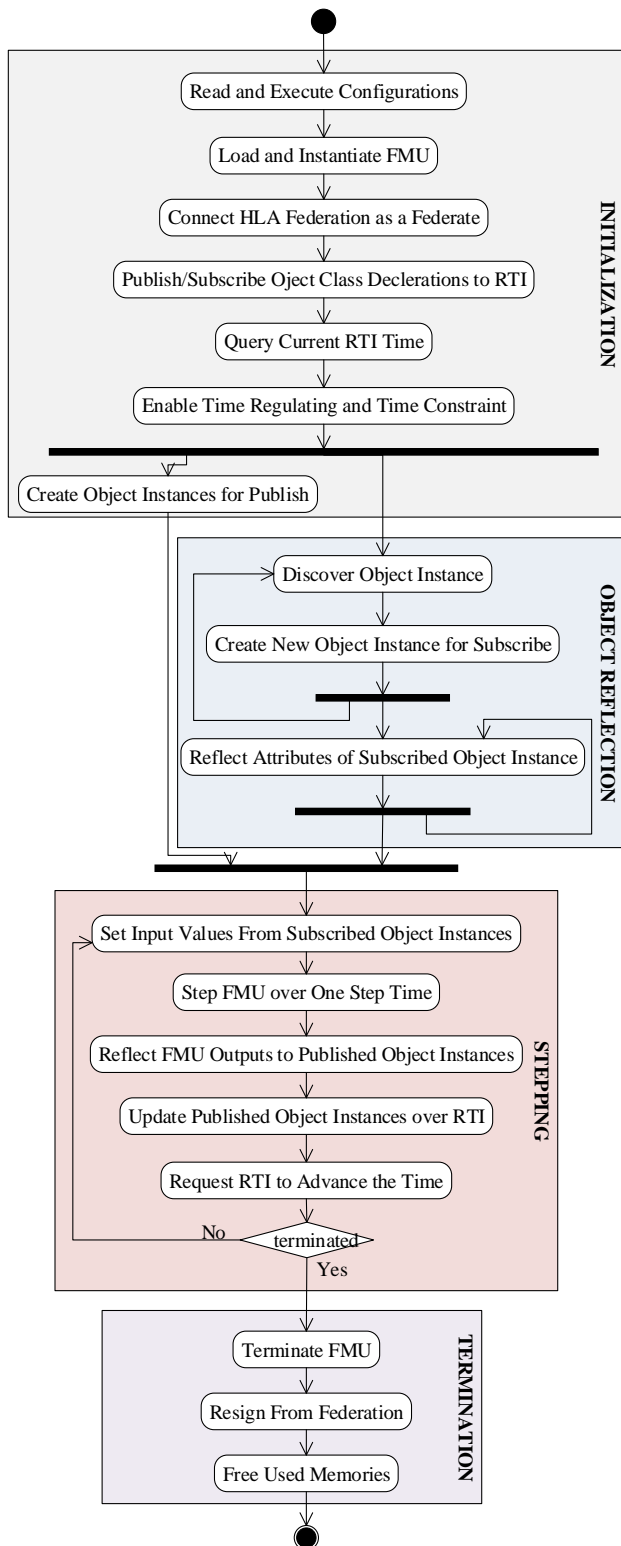


Figure 3– The FMUFd Activity Diagram

The FMU related operations are developed based on the FMU SDK [11]. By using these operations, FMUFd can load and use the FMU. The shared library, inside the FMU file should supply FMI Co-Simulation interface implementations. FMUFd loads

those implementations automatically and then instantiates the model and gets the model instance. By then, FMU is ready to run steps over time.

### 4.2 FMU as an HLA Federate

This section describes how the FMUFd can join a federation execution as a member federate.

#### 4.2.1 Connect to the HLA Federation

The FOM file contains all data exchange related information of HLA Federation, including object classes and object class attributes. By using this file, the FMUFd identifies the structure of each object class with attributes and data types.

The parsing process of a FOM takes two steps. First of all, the data types are parsed and stored in a map. For each type of the data, different parsing procedure is applied as each type has its special fields. For example, the size and endian information is set only for basic data types. Then, the object classes are parsed with their attributes and the data type of each attribute is retrieved from the map. If a class is derived, then its inherited attributes are obtained from its ancestors.

After parsing the FOM file, FMUFd tries to connect to the federation. The federation information is provided by the user through a configuration file. The FMUFd reads this file to get the federation name, path of FOM file and the name of its own federate. Then, FMUFd tries to create a federation if it has not been created yet. Finally, it joins the federation.

After joining the federation, FMUFd declares RTI which object classes with which attributes will be published and/or subscribed. This information is provided by the user with a SOM file. The FMUFd reads the file and identifies the published/subscribed objects and informs the RTI.

#### 4.2.2 Create Object Instances

There are two scenarios for creating the object instances. At the beginning of the simulation, after declaring the object classes, the FMUFd creates the object class instances for publishing the FMU output parameters. The initial values of this object can be assigned by user with using the configuration files. Then, whenever an object class is discovered (new object instance is subscribed), the FMUFd creates an instance of the discovered object class.

Each object contains both object class metadata and attributes. Each attribute allocates the memory with the same size as its data type. While calculating the size of a data type the padding rules are used as described in section 3.3. Although there exists some

rules, still it may not be straightforward to find the exact size of the data type. For example, fixed record data type can contain another fixed record data type and a dynamic array data type. In this case, it is not possible to find exact size of the data type without filling the exact data. Therefore, for every update, the size of the data type should be recalculated. This recalculation may have a problem regarding the performance of an application. To address this issue, the FMUFD has been designed with two restrictions:

- The array data type with dynamic cardinality is not supported by FMUFD,
- The discriminant value of the variant record data type is explicitly defined in configuration file and cannot be changed in runtime.

With these restrictions, the FMUFD calculates the size of each attribute at the beginning of the simulation and uses this size throughout the simulation. As the data can contain different data types in it, the calculation may be performed through recursion. The basic data type is the only type with known size. Whenever the recursion reaches a basic data type, the padding rules are applied. The base case of the recursion could be the code segment given in Figure 4. The *currentOffset* value is passed into recursion which holds the previously calculated offset. After recursion is finished, *currentOffset* will hold the size of the root data type.

```

if(theDataType->type ==
ObjectClass::Attribute::DataType::BasicData)
{
    int mod = theDataType->size;
    int padding = (theDataType->size
                  - (currentOffset % mod))
                  % mod;
    theDataType->offset = currentOffset + padding;
    currentOffset = newDataType->size
                  + theDataType->offset;
}

```

Figure 4 – The base condition code snapshot for calculating the padding bytes

#### 4.2.3 Update/Reflect Object Class Attribute

A complex data type can contain both big endian and little endian data types in it, independent from application computer's endian type. Therefore, before updating the object class attribute, the attribute values should be encoded to the right type of endian. Likewise, after reflecting the attribute, the value should be decoded to the computer endian type. The FMUFD always keeps the data with the same encoding of computer. By doing that, it becomes easier to use the data in an application. Whenever an attribute is needed to be updated, the attribute is encoded first then update

operation is called. Likewise, whenever an attribute is reflected, the value of that attribute is decoded first and kept in decoded form in memory.

The encode/decode operation is also executed with recursion. The basic data type is the only type with known endian type. Whenever the recursion reaches to the basic data type, the swapping operation is applied. The base case of the recursion could be the code segment given in Figure 5. The *returnValue* and *rowData* are the void\* data type values, with the same size of attribute. If the recursion is used for updating the attribute operation than *rowData* refers to the current value of the attribute, otherwise, it refers to the reflected value of the attribute. The *returnValue* refers to the encoded (or decoded) value of the attribute.

```

if(dataType.type
== ObjectClass::Attribute
::DataType::DataType::BasicData)
{
    if(currentNodeEndianType
    != dataType.endianType )
    {
        T_UINT8* returnValueOffset
        = (((T_UINT8*) returnValue )
          + dataType.offset);
        T_UINT8* rowDataOffset
        = (((T_UINT8*) rowData )
          + dataType.offset);

        switch(dataType.size)
        {
            case sizeof(T_UINT8):
                *returnValueOffset = *rowDataOffset;
                break;
            case sizeof(T_UINT16):
                *((T_UINT16 *) returnValueOffset)
                = _byteswap_ushort(*((T_UINT16 *) rowDataOffset));
                break;
            case sizeof(T_UINT32):
                *((T_UINT16 *) returnValueOffset)
                = _byteswap_ulong(*((T_UINT16 *) rowDataOffset));
                break;
            case sizeof(T_UINT64):
                *((T_UINT16 *) returnValueOffset)
                = _byteswap_uint64(*((T_UINT16 *) rowDataOffset));
                break;
        }
    }
}

```

Figure 5 – The base condition code snapshot for encoding/decoding the attribute values.

### 4.3 Running the Federate

After introducing how HLA data is de-marshalled, the next step is mapping FMU scalar variables to HLA basic data types. This mapping is performed through user configuration files. These files inform the FMUFD about which data from HLA will be set to FMU and which data from FMU will be published to HLA.

After mapping between FMU scalar variables and the HLA attributes, the stepping function can be executed.

Before running a step of FMU, the FMUFD updates each input variable of the model. The input values are obtained from an instance of related object class. If there is no instance for related object class then the FMUFD will wait for the instance of that related object.

After running a step of FMU, the FMUFD updates related attributes of the HLA objects by retrieving the values from related FMU scalar variables. Therefore, an FMU output values can be mapped to different HLA objects, which are controlled by the FMUFD. After value updates are finished, the FMUFD will request the RTI to publish those attributes.

#### 4.4 FMUFD Structure

The FMUFD is implemented as an application. Inspired from paper [14], the application is constructed with three base layers as Figure 6.

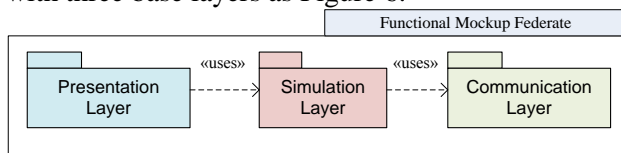


Figure 6 – the FMUFD Structure

##### 4.4.1 Presentation Layer

The presentation layer is the user interface of the application. This layer provides presentation of application, input and interaction with the user as shown in Figure 7. The plot in the figure shows the change of some parameters of the missile and target over time. By using this layer, a user can load the necessary configurations to FMUFD and observe scalar variables' value changes in real time.

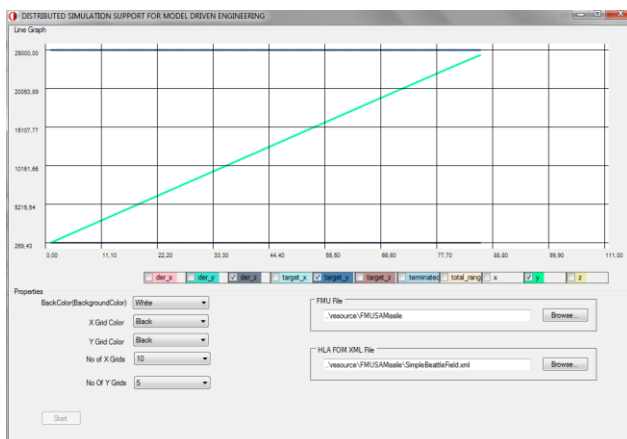


Figure 7 – The FMUFD screenshot while running the missile FMU

##### 4.4.2 Simulation Layer

The simulation layer processes the application. It includes the computation of FMI simulation and federate specific HLA object classes. Its purpose is to run FMU and generate the federate behavior.

Simulation layer is responsible for running the simulation. This layer initializes the FMU, supplies necessary inputs for FMU from HLA class instances, runs the models and publishes the model outputs over HLA distributed environment.

One of the key features of the simulation layer is to create HLA object class structure dynamically. That is, without having the real structure, simulation layer can create a void data with the same size of the structure by using FOM xml file. Then simulation layer can edit this void data parts with the same position of any object class attribute fields.

##### 4.4.3 Communication Layer

The communication layer deals with the RTI communication in order to access the object classes and interactions exchanged in the federation execution. RTI is the middleware that manages the federation execution and object exchange through a federation execution. In addition to data exchange, communication layer also supports time management service.

#### 4.5 FMUFD Capabilities

In this section, FMUFD capabilities are described in terms of HLA interface services. Data distribution management and ownership management are not used in our current implementation.

**Federation Management:** If the federation has not been created before, the FMUFD creates the federation. Then it joins the federation. Similarly, after simulation is finished, the FMUFD resigns from the federation and if there is no other federate connected to the federation, it destroys the federation.

**Declaration Management:** FMUFD informs the RTI about publishing/subscribing object classes with attributes.

**Object Management:** Whenever new object class instance is discovered, FMUFD keeps the handle for this instance and allocates memory for it. Whenever a *reflectAttributeValues* event is raised by RTI, the FMUFD check whether the object instance is discovered before. If it is discovered, FMUFD reflects the attribute values to the allocated memories of the object instance, and ignores otherwise. Whenever a *removeObjectInstance* event is raised by RTI, the FMUFD checks if the object instance is discovered before. If it is discovered, FMUFD deletes the handle of instance and frees the related allocated memory.

FMUfd reflects the attribute values to the allocated memory of the object instance, ignores otherwise.

**Time Management:** FMUfd works as a time regulating and time constraining federate. As the nature of the time constraint, FMUfd ensures that the subscribed object model instance received reflection no less than the *currentRTTime*. Also, after each running step of the model, FMUfd requests to update the federate time.

#### 4.6 Limitations

FMUfd still has some limitations and constraints. First of all, HLA interactions are not supported by FMUfd as there exists no corresponding logical concept in current FMI for Co-Simulation standard. Moreover, the array data type with dynamic cardinality is not supported by FMUfd. Finally, the discriminant value of the variant record data type is defined at the beginning of the simulation and FMUfd does not allow changing it at runtime.

## 5 Demonstration

To demonstrate the FMUfd usage, a simple distributed simulation environment is developed with MAK HLA RTI [12] implementation. For this application, the RPR2-D17 FOM file developed by SISO [13] is used as a FOM file. The used HLA object classes with its parent hierarchy are shown in Figure 8.

```
HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft
HLAobjectRoot.BaseEntity.PhysicalEntity.Munition
```

Figure 8 – RPR2-D17 FOM classes used in the case study.

There are three nodes connected over an Ethernet network in this distributed simulation environment as shown in Figure 9. In the missile node, the missile co-simulation FMU (called *MissileFMUfd*) is connected to distributed simulation environment as the HLA federate by using FMUfd. Similar to missile PC, the aircraft co-simulation FMU is also connected to the simulation environment as a federate by using FMUfd (called *AircraftFMUfd*) in the target aircraft PC. The synthetic environment node is used to provide other entities in this operational setting, such as the missile launch platform, and to visualize the simulation in 2D and 3D. To this end, Presagis STAGE is used [15].

With two configuration files, one for FMU inputs and one for FMU outputs, user should supply information to the FMUfd about mapping between FMU scalar variables and HLA basic data types. Therefore,

the entire hierarchy down to the basic data types should be explicitly defined for an object model.

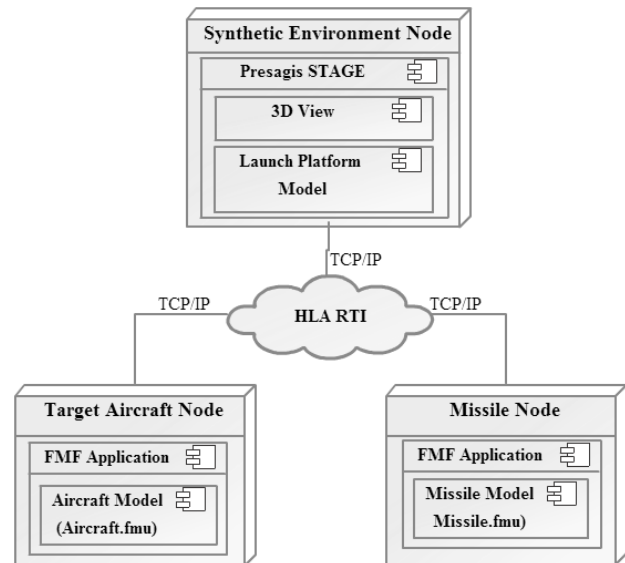


Figure 9 – The Deployment View Diagram of Simulation Environment

The example extract from a configuration file is provided in Figure 10. This example shows how the *Target\_Ecef\_X* scalar variable can be mapped with the *Aircraft* object's *Spatial* attribute's data type where data type goes down the hierarchy until it reaches the basic data type *HLAfloat64BE*. This mapping is specified for other scalar variables as well.

```
Target_Ecef_X = Aircraft|Spatial|SpatialStruct
:DeadReckoningAlgorithm-A-Alternatives
: SpatialStruct-DeadReckoningAlgorithm[DRM_FPW]
: SpatialFPStruct:WorldLocation:WorldLocationStruct:X
:HLAfloat64BEmetersperfectways:HLAfloat64BE
```

Figure 10 – The example mapping between FMU scalar variables and HLA object class attribute data types.

With these configurations, 1500 simulation runs have been executed and performance figures for framework overhead have been measured. The median time for updating FMU parameters from HLA objects for *MissileFMUfd* is 254 microseconds. Likewise, the median time for updating HLA attributes from FMU parameters for *MissileFMUfd* is 356 microseconds. Measurement were taken on a computer with Intel Xeon 2.66GHz processor, 4GB DDR3 RAM and Windows 7 Pro 64bit operation system.

## 6 Conclusion

The FMI is an emerging standard for co-simulation and model exchange in Model Based Integration community. Also, HLA is a well-accepted standard for the distributed simulation. FMUfd supplies a solution for participation of FMUs that implement FMI Co-Simulation interface in an HLA Federation. Thus, a system model can be simulated as a part of an aggregate simulation of its operational setting. Moreover, this promotes a high level of reusability of system models supporting FMI.

As an alternative approach, the wrapping process may generate an FMU HLA wrapper code automatically by reading the FMU specification and generating the wrapper that knows how to translate just the specific FMU join HLA. This approach, on the other hand, is both FMU and federation specific and requires a recompile for each FMU. In our case, we aimed at recompilation free integration of FMUs to any federation via configuration files.

Finally, FMUfd is currently released in Roketsan Inc. as an in-house developed simulation infrastructure and being employed in some system development projects.

## Acknowledgments

This work is supported by Turkish Ministry of National Defense, Undersecretariat for Defense Industries [Project Name: MOKA].

## References

- [1] HLA Working Group of IEEE: IEEE Std 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. New York, USA, 2000.
- [2] HLA Working Group of IEEE: IEEE Std 1516.1-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. New York, USA, 2000.
- [3] HLA Working Group of IEEE: IEEE Std 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification. New York, USA, 2000.
- [4] MODELISAR Consortium: Functional Mock-up Interface for Model Exchange Version 1.0, [www.functional-mockupinterface.org](http://www.functional-mockupinterface.org), January, 2010
- [5] MODELISAR Consortium: Functional Mock-up Interface for Co-Simulation Version 1.0, [www.functional-mockupinterface.org](http://www.functional-mockupinterface.org), October, 2010
- [6] Stenzel, C.: Distributed Simulation in Technical Applications, X International PhD Workshop, OWD 2008, Conference Archives PETiS, Vol. 25, 2008, Gliwice, Poland, 513-518
- [7] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C. and Derler, P.: Distributed Simulation of Heterogeneous and Real-time Systems. (2013) In: 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications - IEEE/ACM DS-RT 2013, 30 October 2013 - 01 November 2013 (Delft, Netherlands).
- [8] Stenzel, C., Pawletta, S.: CERTI - Bindings to Matlab and Fortran, University of Wismar, <http://www.mb.hs-wismar.de/~stenzel/software/MatlabHLA.html> Germany, 2008 (Last Accessed 20/11/2013)
- [9] HLA Toolbox <sup>TM</sup> The MATLAB® interface to HLA, <http://www.forwardsim.com>
- [10] Awais, M. U., Palensky, P., Elsheikh, A., Widl, E. and Matthias, S.: The High Level Architecture RTI as a master to the Functional Mock-up Interface components. Vienna, AUSTRIA, 2013
- [11] QTronic Company Developer Team: FMU SDK version 1.0.2, <http://www.qtronic.de/en/fmusdk.html>, 2010
- [12] MÄK Technologies: MAK RTI User's Guide, 2013
- [13] Shanks, G.: Real-time Platform Reference Federation Object Model (RPR FOM) Version 2.0D17, Simulation Interoperability Standards Organization, 2003



- [14] Topçu, O. and Oğuztüzün, H.: Layered simulation architecture: A practical approach", Simulation Modelling Practice and Theory (SIMPAT) Journal, vol. 32, March 2013, pp. 1-14.
- [15] Stage Sales Team: Stage, A Complete Simulation Development Environment, <http://www.presagis.com>, Montreal, CANADA, 2013



# Transmission Modeling in Modelica: A consistent approach for several software development platforms

Jochen Köhler, Michael Kübler, Julian King  
ZF Friedrichshafen AG  
Graf-von-Soden-Platz 1, D-88046 Friedrichshafen, Germany  
{jochen.koehler, michael.kuebler, julian.king}@zf.com

## Abstract

Simulation models play a fundamental role in the development of transmission control software. In the ideal case, the same model can be used throughout the whole development process from concept and design over implementation to system verification. The idea is to use one uniform model along this V-scheme. This leads to the requirement that simulations have to be able to run in real-time on hardware-in-the-loop platforms. On the other hand, very detailed models of some components might be needed during the early design phase.

Thus, a trade-off between modeling depth and computational performance has to be found. This may be achieved by selectively simplifying parts of the model that are prone to generating stiff sub-systems or a large number of state events.

Within this framework, the present paper introduces the Modelica simulation model of TraXon, the new modular transmission for heavy commercial vehicles by ZF. The model can be adapted to various needs by replacing components according to the required modeling depth and/or dynamical behavior.

After a brief overview of the ZF in-house Modelica libraries and the architecture of the TraXon model, some approaches and tools are described for evaluating and optimizing models with respect to real-time issues.

*Keywords: ZF Modelica libraries; model simplification; performance analysis; hardware-in-the-loop;*

## 1 Introduction

ZF is a global player in driveline and chassis technology. Being confronted with a huge variety of increasingly complex transmission concepts and continuously reduced development cycle times, simulation has traditionally been an essential part of transmission control software engineering at ZF. Detailed

models reflecting the mechanical structure of the transmission as well as the physical behavior of its actuating components are central to this task.

This paper outlines the Modelica simulation model of the new ZF TraXon transmission. Here, the principles of object-orientated modeling inherent to Modelica greatly contribute towards balancing both modeling accuracy and real-time performance. Thus, one simulation model can be used throughout the whole development process.

## 2 Usage of Modelica in ZF

### 2.1 Motivation

Modelica was introduced at ZF over 10 years ago. The main intention was to standardize the modeling methodology for transmission systems and to share modeling know-how by creating component libraries accessible throughout the company [1].

Modelica nowadays represents the standard approach for modeling and simulating of a wide range of distinct transmission systems, particularly in the context of control algorithm development. Originally starting from one single ZF Modelica Library, more than ten different context-specific libraries have been developed so far, all based on the Modelica Standard Library [2].

### 2.2 ZF Modelica Libraries

While some of the above mentioned Modelica libraries are only employed for targeted investigations regarding, e.g., shifting comfort or combustion engine dynamics, others are regularly used in almost every transmission model formulation:

- ZFLib, encompassing central gear parts such as clutches, synchronizers, table-based engine formulations, controllers, chassis models, etc. It is based on the Modelica Standard Library and also con-

tains some extensions to the components therein.

- HybridLib, including extensions for hybrid powertrains (electrical machines, battery models and complete driving strategies for hybrid vehicles).
- FluidLib for pneumatic/hydraulic actuating cylinders and valves. This library was built upon Modelica.Media and Modelica.Fluid. The separation of fluid-mechanic components and the employed media is quite useful in order to design models that can be used for both hydraulics and pneumatics.

The motivation for developing in-house libraries rather than using commercial ones stems from the following requirements:

- Adaptability of components according to internal standards, e.g., with respect to data handling and parameterization [1].
- Preservation of ZF-specific modeling know-how inside components.
- Synergy effects when modeling closely related transmission concepts.
- Models perfectly tailored to the needs of ZF.

In particular, the central libraries are widely reused among distinct transmission modeling projects, thus leading to thoroughly tested components characterized by a high degree of reliability and robustness. Dymola [3] represents the standard front-end tool. However, as Modelica is tool-independent current efforts also aim at evaluating the usability of the ZF Modelica libraries in SimulationX [4]. Then, users may choose the most suitable tool for their task.

### 2.3 Model Export

In addition to the unified modeling approach described above, also a standardized export process has been implemented. The possibility to make simulation models available on various software design and testing platforms is an essential requirement. One common software development and integration tool is Simulink [5] with its ability to export models to other platforms.

For this purpose, the Dymola-Simulink interface has been adapted in order to serve all simulation environments used at ZF, especially ZF-internal software-in-the-loop and hardware-in-the-loop platforms. One particular extension to the original interface is that different fixed-step solvers are attached to the exported models. These solvers can be selected just before running the simulation, so the user may decide whether he wants to use the Simulink

solver or one of the built-in fixed-step integrators. Furthermore, the inner simulation time step can be defined independently from the Simulink solver step size. If the inner step size is smaller than the global step size a sample and hold mechanism is implemented for in- and outputs.

## 3 The TraXon simulation model

### 3.1 TraXon – the new modular transmission by ZF

TraXon (Figure 1) is an automatic commercial vehicle transmission platform with five modules for different driving applications, satisfying the most challenging requirements. It is prepared for intelligent networking with other vehicle systems, and sets new standards in the areas of efficiency, comfort, and application diversity [6].

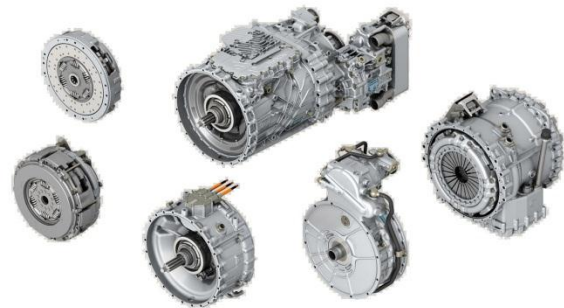


Figure 1: TraXon transmission

### 3.2 TraXon simulation model: overview

The modular structure of the TraXon transmission is also reflected in the structure of the simulation model. Correspondingly, the TraXon gearbox model consists of six mechanical sub-systems in series, each one associated with its own (pneumatic) actuator concept (Figure 2). In particular, these sub-systems all have a uniform connector interface and can thus easily be rearranged to model a wide spectrum of distinct transmission topologies in a consistent manner.

Figure 3 exemplifies the general structure of such a sub-system (splitter). Note that there is a clear separation between the rotational part, mimicking the torque flow through a synchronizer depending on the position of the sleeve, and the actuator, which trans-

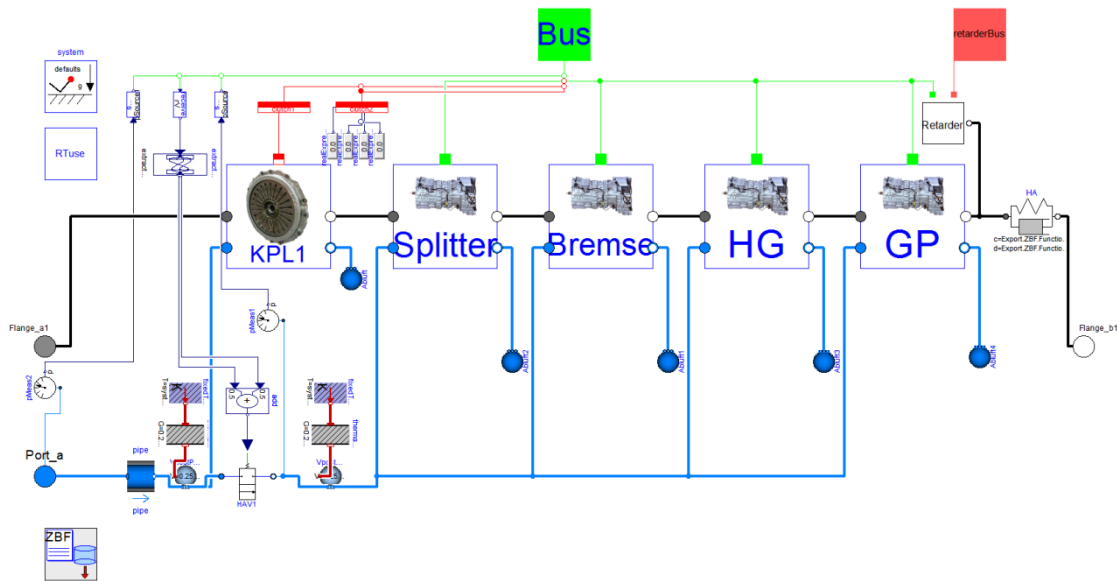


Figure 2: TraXon simulation model: engine clutch, splitter (synchronizer), brake, main gear (dog clutch transmission with specified shifting geometry), group transmission (synchronizer) and retarder

lationally moves the sleeve according to the pressure relationships inside a two-position pneumatic cylinder.

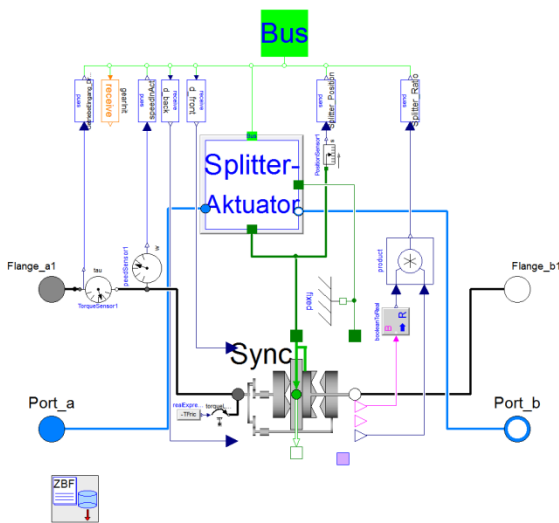


Figure 3: Splitter part of the TraXon transmission, modeling a synchronizer actuated by a pneumatic cylinder

By extending from a common interface, different actuator concepts (e.g., electro-magnetic or electro-mechanical) and actuator formulations with varying modeling depth can thus quickly be exchanged and simulated. This is especially important when the model needs to be simplified for real-time simulations.

### 3.3 Detailed modeling approach

The master model contains all relevant physical effects of the pneumatic actuators and can be employed for detailed MiL/SiL-investigations during function development.

#### 3.3.1 Detailed actuator cylinder model

Almost all components of TraXon are actuated by pneumatic cylinders due to the fact that this kind of energy is already available on heavy trucks. The physical model of these actuator cylinders consists of replaceable variants of chamber and piston components. The most detailed piston model considers friction with stick-slip-effects. This results in high calculation effort, so the piston may also be replaced (using the “replaceable” mechanism of Modelica) by other variants building on continuous friction modeling approaches. End-stops are generally not taken into account; instead they are incorporated into the translational dynamics of actuated shifting device. This helps to reduce the number of equations without any effect on the overall behavior of the model. The detent of the piston is also modeled, assuring that the piston stays in a pre-defined position if not actuated.

Chambers are composed of a fluid volume and a translational flange, transmitting force according to the product of fluid pressure and plunger area. The latter may be constant or position-dependent as in the case of “three-position” cylinders, where the piston can be held at a center-position.

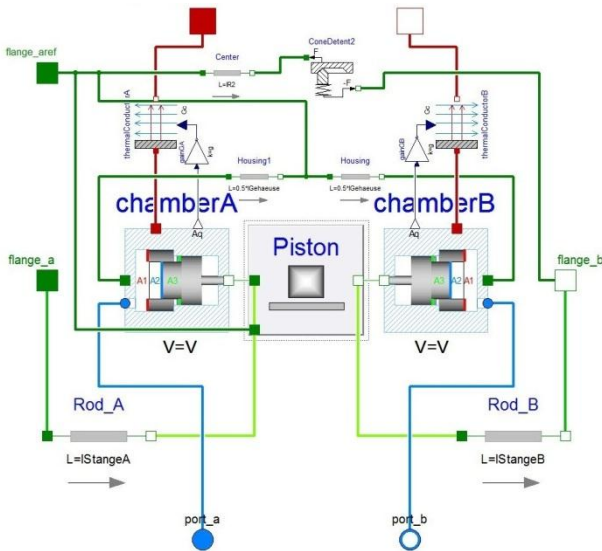


Figure 4: Detailed cylinder model using replaceable components

A major advantage here is that the modeling media are separated from the components so that the cylinder models can be used for pneumatic and also for hydraulic systems just by changing the media.

### 3.3.2 Detailed actuator valve model

The actuator valve models governing the venting behavior of the respective cylinders are divided into an electrical, mechanical and fluid-dynamic part.

The electrical part contains coil inductance and electro-magnetic force. Control signals are generated by Pulse-Width-Modulation. Simplifications are made in the magnetic part by neglecting hysteresis and using tables instead. The mechanical part consists of a translational spring-mass system for the valve tappet. End-stops are modelled and friction can be activated but is not used at the moment.

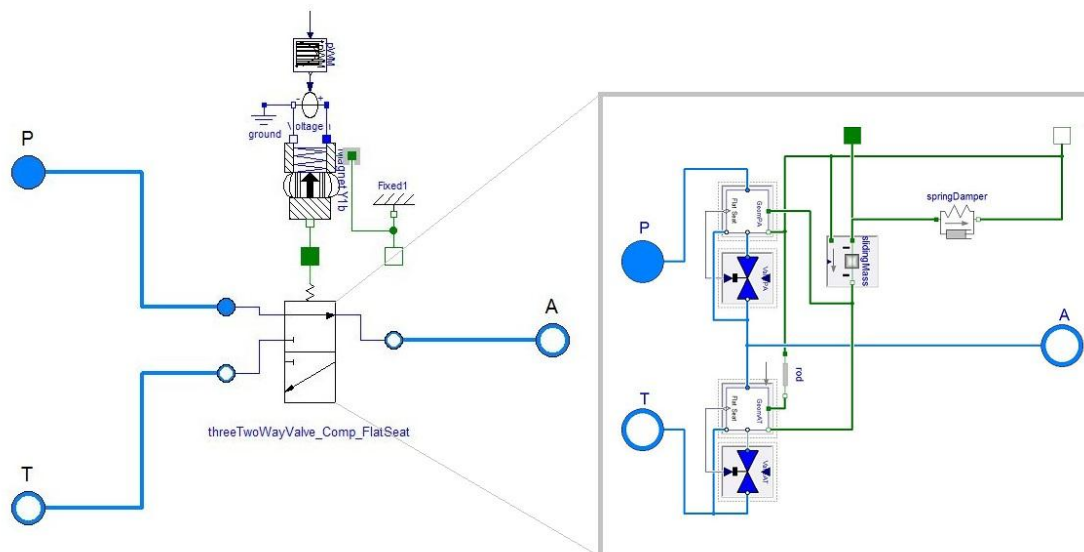


Figure 5: Detailed valve model composed of electrical, mechanical and fluid-dynamic part

The fluid-dynamic part is divided into a geometrical and a fluid flow part, both of which are replaceable. The geometrical part yields the valve opening resulting from a specific position of the tappet. Different valve seating geometries can be realized, including flat seat and ball-drill/-cone seat. The fluid flow part calculates mass flow depending on the valve opening.

Here, changing from pneumatics to hydraulics is almost as easy as in the case of cylinders. The only component that has to be replaced is the inner valve containing the equations to calculate the pressure drop. This is due to the fact that a different behavior results dependent on the compressibility of the media (compressible (e.g., air) vs. incompressible (e.g., simple oil)).

### 3.4 Simplifications made for real-time computations

Since the TraXon simulation model aims at accompanying the entire software development process, there is a special constraint: real-time capability.

To meet this challenge, certain model simplifications have to be implemented in order to guarantee a fast and robust simulation on HiL-platforms, usually employing an Euler-forward algorithm with a fixed step size of 1 ms.

In this context, extensive use is made of Modelica's inherent feature to make certain model components easily exchangeable. In particular, by declaring the above-mentioned actuator parts as replaceable, cylinder and valve formulations with different levels of detail may directly be incorporated into the model without changing the surrounding structure. This feature is especially beneficial during model simplification steps as outlined below.

### 3.4.1 Simplified actuator cylinder model

In the simplified cylinder model the plunger areas are parameterized such that the neutral position corresponds to a force equilibrium if both chambers have ambient pressure. Therefore, the detent force can usually be neglected. Friction is modeled in a velocity-dependent way in order to avoid stick-slip-events and thus the creation of many event iterations. The volumes of the in- and outgoing pipes are lumped into the dead volume of the chambers. This avoids fast dynamics due to small fluid volumes and hence leads to better numerical stability.

### 3.4.2 Simplified actuator valve model

In contrast to the detailed valve model the electro-mechanical part is replaced by a second-order transfer function. Thus, higher frequencies of the valve piston movement can be avoided. Also the generation of events is reduced by neglecting end-stops of the piston.

## 3.5 Stability analysis

If simulation has to be performed with a fixed step solver, the stability region of the integrator has to be taken into account. Facing a complex model like the one introduced before, with multiple domains interacting with each other, it is difficult to identify the critical modes of the system, i.e., those modes that do not fulfill the stability requirements because their eigen-frequencies are too fast.

To overcome this problem a MATLAB [5] tool was implemented, which visualizes all eigenmodes and eigenvectors that can be computed by linearizing the nonlinear system at certain time points of interest.

The linearizing feature of Dymola is used for this purpose. The top panel of Figure 6 shows the stability regions for the implemented fixed step solvers.

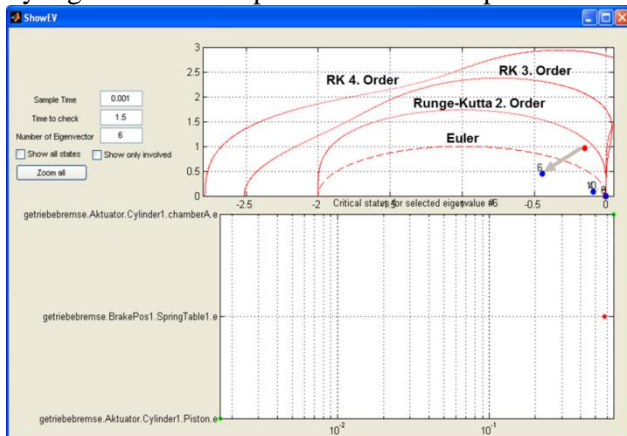


Figure 6: Eigenmode analysis tool

The blue dots show all eigenmodes in the z-space. Each dot that does not fall within the stability region of the used solver has to be investigated. At the bottom of Figure 6, the eigenvector of one selected eigenmode is shown.

This links the eigenmode to the contributing state variables and thus allows for identifying the corresponding components in the model. To enhance this feature, special states are included in all components, representing the stored energy. The eigenvectors can be “filtered” according to these energy states, making it even easier to find the components of interest. With this information, the engineer may decide to change the model structurally or to adapt some physical parameters, in order to move the eigenmodes into the stability region without modifying the dynamical behavior too much.

## 3.6 Performance analysis

A model analysis after symbolic manipulation of the underlying equations shows the differences between the detailed and simplified modeling approach as discussed above, see Table 1. Note that the number of mixed real/discrete systems of equations is drastically reduced, hence less computational effort is expected during event iterations.

	Detailed model	Simplified model
Number of mixed real/discrete systems of equations	20	4
Number of states	180	124
Number of linear systems of equations	4	4
Max. size of linear systems of equations	8	8
Number of nonlin. systems of equations	12	7
Max. size of nonlin. systems of equations	17	17

Table 1: Model analysis before and after simplification

The performance increase by using the simplified modeling approach can be seen in Figure 7. This computational load can be made visible because the translated model is running in our own simulation framework by calling the “dsblock” C-function [7], and the time needed for its execution is measured at each time step. A very important effect that can be seen in this plot is the occurrence of events and the time needed to handle them. In general, finding consistent restarting conditions after an event is a serious problem for real-time simulations as it seems that at every event handling the calculation time required is at least two times higher than the average time without an event. Due to this effect the maximum al-

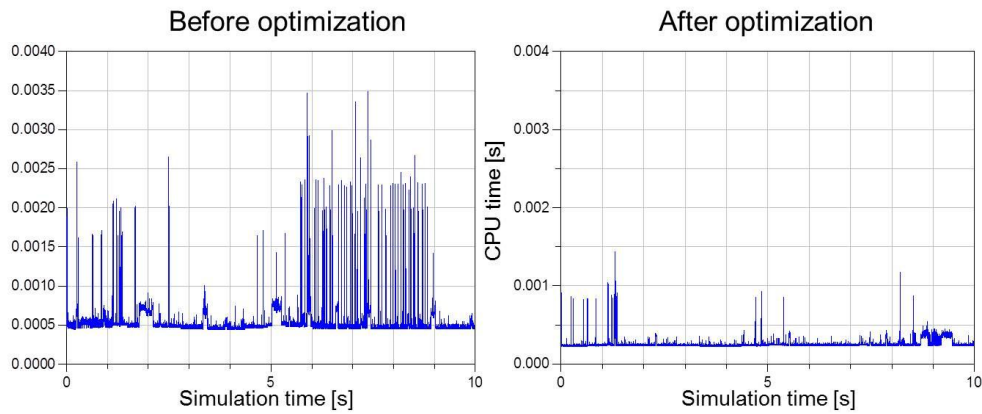


Figure 7: Computational performance before and after simplification

lowed calculation effort is half the integration step size – here 0.5 ms – to avoid time overruns.

Using this analysis tool, the average time interval needed for executing one solver step can be lowered by approximately 50%. Another important effect is the reduction of state events and the parallel decrease of iteration steps necessary for finding consistent restart conditions. When using the TraXon simulation model on hardware-in-the-loop platforms this is the key to avoid overruns.

## 4 Conclusions

The use of simulation models along the entire software development process is widely implemented at ZF. A standardized way to export models to different simulation platforms has been established and the potential of the Functional Mockup Interface in this regard is currently evaluated.

Modelers can benefit from the availability of in-house Modelica libraries and components that are extensively reused in many simulation models.

For real-time applications, object oriented modeling provides an elegant way to rapidly switch between different model formulations with a varying degree of detail. This is a key feature when the model has to be simplified in order to ensure a fast and robust execution. Nevertheless, the analysis methods inside Dymola with respect to this task may be improved. In particular, an automated way for ranking model variables according to their contribution to different timescales in the model would be highly desirable.

Furthermore, the generation of events and the subsequent iteration process represents a drawback of Modelica in the context of for real-time applications.

## References

- [1] Koehler J, Banerjee A. Usage of Modelica for transmission simulation in ZF. Proceedings of the 4th Modelica Conference 2005, Hamburg, Germany, Modelica Association, 7-8 March 2005
- [2] Modelica Standard Library (Version 3.2), Modelica Association, 2010, <https://www.modelica.org/libraries>
- [3] Dymola 2014, Dassault Systemes <http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola>
- [4] SimulationX 3.6, ITI GmbH, <http://www.itisim.com/simulationx/>
- [5] Matlab / Simulink, The Mathworks, <http://mathworks.com>
- [6] ZF Friedrichshafen AG, [www.zf.com/TraXon](http://www.zf.com/TraXon)
- [7] The DSblock model interface, <https://www.modelica.org/documents/DSblock>



# Vectorized single-track model in Modelica for articulated vehicles with arbitrary number of units and axles

Peter Sundström<sup>1</sup>, Bengt Jacobson<sup>2</sup>, and Leo Laine<sup>3</sup>

<sup>1</sup>Modelon AB

<sup>2</sup>Chalmers University of Technology

<sup>3</sup>Volvo Group Trucks Technology

## Abstract

A linear single-track model for articulated vehicles has been implemented. The model can represent an articulated vehicle with an arbitrary number of units each with an arbitrary number of axles. Lateral and yaw dynamics are in focus but longitudinal effects in couplings are also included. In the model, tire forces are linear with respect to slip angle. The couplings between units are represented as non-linear kinematic constraints which are valid for small and large articulation angles.

Four use cases are presented: Inverse dynamics for feedforward control, frequency responses when varying parameters, steady-state evaluations and dynamic simulation. For these use cases, four parametrizations of the model are used corresponding to a tractor with a semitrailer a truck with a dolly and a semitrailer, an A-double (tractor+semitrailer+dolly+semitrailer) and an approximate version of an airport baggage carrier with five full trailers.

*Keywords: vehicle dynamics, vehicle models, articulated vehicles*

## 1 Introduction

Simple vehicle dynamics models are very useful for basic analysis, rough parameter tuning and concept studies. By keeping models simple, one gains not only simulation speed but the possibility to for example invert the model for feedforward control. A simple model is however not always simple to model.

Several modeling approaches have been used to model articulated vehicles [1], [2], [3], [4]. For simple models, it is common to simplify the coupling equations using small angle approximations. This greatly simplifies the modeling effort but also invalidates the

model for low-speed cases when articulation angles typically exceed the validity range of such an approximation. For multiple units, keeping the non-linear coupling equations usually require some kind of symbolic solver, such as Maple, Mathematica or Matlab's Symbolic toolbox. It is also common that the modeling is done for a specific vehicle combination with little room for extending to more units without reformulating the model.

Creating the model using Modelica one does not have to explicitly solve all equations and thus the model formulation can be kept simple while letting the Modelica tool take care of equation solving. Furthermore, by vectorizing the model with respect to the number of units and axles the model equations does not have to be expressed with a specific vehicle combination in mind, leaving that to the specific implementation.

## 2 Model description

A description of the model equations and structure is presented here. For brevity and readability of code snippets, `vector()` and `matrix()` function calls and some full library paths are left out. `MSL` is used when models are taken from the Modelica Standard Library, mainly from `Modelica.SIunits` and `Modelica.Blocks`.

### 2.1 Assumptions

As is normal for single-track models, the two tires on an axle are lumped as a single tire in the middle of the axle. Coupling points are assumed to not transfer torque between units. Slip angles are assumed to be small such that  $\tan \alpha \approx \alpha$ .

While the model accepts a variable input velocity,

typical longitudinal effects such as load transfer are not taken into account.

## 2.2 Parameter and variable structure

Parameters describing positions on the vehicle are all expressed relative to the first axle of the corresponding unit. This includes axle positions, center of gravity, c.g., positions and coupling point positions. Figure 1 shows the position parameters for a unit. Since states

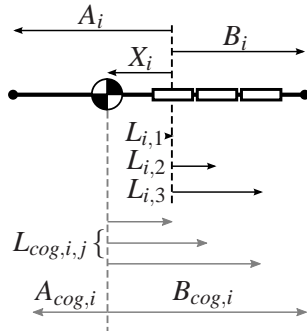


Figure 1: Geometric parameters of unit  $i$ .  $A_i$  and  $B_i$  are the distances from the first axle to the front and rear coupling points, respectively.  $X_i$  is the distance from the first axle to center of gravity.  $L_{i,j}$  is the distance from axle 1 to axle  $j$  on unit  $i$ .

are defined at the center of gravity of each unit, position parameters are recalculated relative to this, e.g.  $A_{cog} = A - X$ .

Parameters and quantities related to axles, such as tire forces and axle positions, are defined as matrices with dimensions  $[nu, na]$  where  $nu$  is the number of units in the combination and  $na$  denotes the maximum number of axles on any of the units. If the number of axles on a unit is less than  $na$  the unused elements will be set to zero and are thus disregarded. Parameters and quantities related to units, such as motion states and masses, are defined as vectors of length  $nu$ .

## 2.3 Tire forces

Given the small angles assumption, the slip angle for a single tire is defined as

$$\alpha = \frac{v_y + L_{cog}\omega_z}{v_x} - \delta \quad (1)$$

where  $L_{cog}$  is the distance from the center of gravity to the axle that is being considered,  $v_y$  and  $\omega_z$  are the lateral velocity and yaw rate of the unit in question and  $\delta$  is the steering angle. In the model code, the slip angle matrix for all axles is defined as

---

```
alpha = ((vy*ones(1,na)
+Lcog.*(wz*ones(1,na)))
./ (vx*ones(1,na))-delta);
```

---

The lateral tire forces can then be calculated as

---

```
Fyw = -C.*alpha;
```

---

where  $C$  is the axle cornering stiffness matrix. Together with the longitudinal forces,  $F_{xw}$ , the tire forces can be transformed to the vehicle coordinate system with the steering angle  $\delta$

---

```
Fx = Fxw.*cos(delta)-Fyw.*sin(delta);
Fy = Fxw.*sin(delta)+Fyw.*cos(delta);
```

---

A variable  $F_{xd}$  is defined as the drive force needed at each driven axle to maintain the input velocity. This force is applied at each driven axle:

---

```
for i in 1:nu loop
  for j in 1:na loop
    if driven[i,j] then
      Fxw[i,j]=Fxd;
    else
      Fxw[i,j]=0;
    end if;
  end for;
end for;
```

---

This force is implicitly determined to satisfy the rest of the model equations.

## 2.4 Coupling constraints

The constraints in the couplings state that the global velocity vector of the rear coupling on the unit in front should be the same as that of the front coupling point on the unit behind. The Modelica language allows for a series of equations to be defined using for loops. Here, the constraint that the velocities at the front coupling of unit  $i$  should be the same as that of the rear coupling on unit  $i-1$  is defined by looping over all the couplings:

---

```
for i in 1:nu-1 loop
  vx[i+1] = vx[i]*cos(theta[i])
  -(vy[i]+Bcog[i]*wz[i])*sin(theta[i]);
  vy[i+1]+Acog[i+1]*wz[i+1]=
  (vy[i]+Bcog[i]*wz[i])*cos(theta[i])
  +vx[i]*sin(theta[i]);
end for;
```

---

Equations for the coupling forces are not formulated explicitly, the velocity constraints in the coupling to-

gether with force and moment balances gives enough information for an implicit calculation. Figure 2 shows how the coupling cut forces are applied to the different units affected by the coupling. The forces

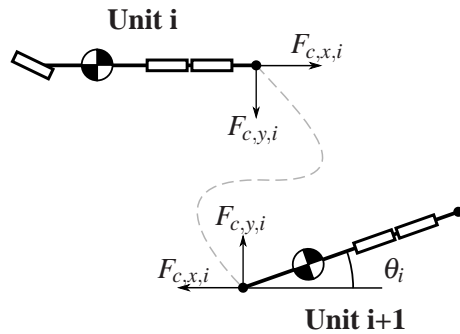


Figure 2: Example of cut forces in coupling between unit  $i$  (front) and  $i+1$  (rear)

in each coupling are defined in the coordinate system of the front unit which it affects with negative sign. The rear unit in the coupling is affected by the forces with positive sign transformed by the articulation angle. So  $F_{cx}[i]$  pulls rearward on unit  $i$  and  $F_{cx}[i] \cdot \cos(\theta_i)$  pulls forward on unit  $i+1$ . Similarly,  $F_{cy}[i]$  pulls in the negative  $y$  direction (rightwards) on unit  $i$  and  $F_{cy}[i] \cdot \cos(\theta_i)$  pulls towards the left on unit  $i+1$ .

## 2.5 Steady state mode

To simplify steady-state analysis, substitute parameters are defined for each state derivative:

---

```
MSL.Acceleration[nu] d_vx;
MSL.Acceleration[nu] d_vy;
MSL.AngularAcceleration[nu] d_wz;
MSL.AngularVelocity[nu-1] d_theta;
```

---

If the model is to be run in steady-state, all derivatives are set to zero. For dynamic simulations they are set to the derivative of their corresponding state variable:

---

```
if steadystate then
  d_vx=zeros(nu);
  d_vy=zeros(nu);
  d_wz=zeros(nu);
  d_theta=zeros(nu-1);
else
  d_vx=der(vx);
  d_vy=der(vy);
  d_wz=der(wz);
  d_theta=der(theta);
end if;
```

---

## 2.6 State equations

By using matrices and vectors, the state equations can be formulated as matrix equations for the entire vehicle combination instead of writing separate equations for each unit. In the axle force matrices, a row,  $i$ , contains all the axle forces on unit  $i$ . To get the force sum on each unit, the force matrix is multiplied by a column vector of ones of length  $n_a$ :

$$\begin{bmatrix} F_{y,11} & \cdots & F_{y,1n_a} \\ \vdots & \ddots & \vdots \\ F_{y,n_u1} & \cdots & F_{y,n_un_a} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} F_{y,11} + \dots + F_{y,1n_a} \\ \vdots \\ F_{y,n_u1} + \dots + F_{y,n_un_a} \end{bmatrix} \quad (2)$$

For coupling forces column vectors  $[F_{cx}; 0]$  and  $[F_{cy}; 0]$  are defined for the rear coupling forces, the 0 element meaning no rear coupling force on the rear-most unit. Similarly, vectors  $[0; F_{cx}]$  and  $[0; F_{cy}]$  are defined for the front coupling forces where 0 here means no front coupling force on the first unit. Then, the force balance equations for the whole combination can be written as a matrix equation using element-wise multiplication. The lateral and longitudinal force balances are written as:

---

```
ay=d_vy+vx.*wz;
m.*ay=Fy*ones(na,1)-[Fcy;0]
+[0;Fcx].*sin([0;theta])
+[0;Fcy].*cos([0;theta]);
```

---



---

```
ax=d_vx-vy.*wz;
m.*ax=Fx*ones(na,1)-[Fcx;0]
+[0;Fcx].*cos([0;theta])
-[0;Fcy].*sin([0;theta]);
```

---

The moment balance around the yaw axis is done in the same way with the added moment arms for the different forces:

---

```
I.*d_wz=Lcog.*Fy*ones(na,1)
-Bcog.*[Fcy;0]
+Acog.*([0;Fcx].*sin([0;theta])
+[0;Fcy].*cos([0;theta]));
```

---

where  $A_{cog}$  and  $B_{cog}$  are the distance from c.g. to the front and rear couplings, respectively and  $L_{cog}$  is the distance from c.g. to each axle.

## 2.7 Inputs

The inputs to the model are the steering angles at all axles and the velocity of the first unit.

---

```
input MSL.RealInput[nu,na] delta_in;
input MSL.RealInput vx_in;
```

---

Due to the kinematic constraints in the coupling, the derivative of the  $v_{x\_in}$  input will be needed. To allow velocity as input when exporting the model, the derivative of  $v_{x\_in}$  is set to zero with annotated derivatives using a QuasiStatic function from the Modelon library:

```
vx[1] =
    max(0.1, QuasiStatic.scalar(vx_in));
```

The output from the QuasiStatic.scalar(vx\_in) function has the same value as vx\_in but with zero derivative. Here, the velocity is also set to be minimum 0.1 to protect sideslip calculations from division by zero.

### 3 Use cases

Four use cases are presented as examples. For each use case a different vehicle combination is used to show how the model can be parametrized for different vehicles.

A tractor-semitrailer model is inverted for feedforward control, a truck-dolly-semitrailer is linearized and its frequency response is studied, a tractor-semitrailer-dolly-semitrailer is run in steady-state to evaluate its off-tracking characteristics and a five-trailer airport baggage carrier is run in a dynamic simulation. Figure 3 shows the four vehicles used.

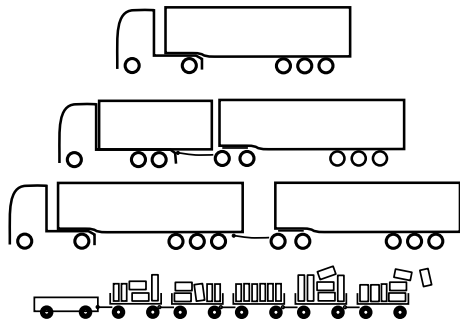


Figure 3: The four vehicle combinations used.

The Functional Mockup Interface, FMI, allows modeling and simulation to be performed in separate tools. Model development has been done in Dymola and all simulations and plotting are done in Matlab/Simulink using FMI Toolbox. The only parameters that need to be set before exporting models are the steadystate setting and the number of units, nu, and axles, na, as they change the structure of the model. Other parameters such as axle positions and cornering stiffnesses are free to set when using the exported model.

### 3.1 Inverse dynamics

By using the InverseConstraints block in the Modelica Standard Library, the inverse dynamics of the model can be solved for. To use the model for feedforward control, we select the lateral acceleration  $a_y$  of the first unit as input and the steering angle at the front axle  $\delta_{out}$  as output. Figure 4 shows the block diagram of the inverse model.

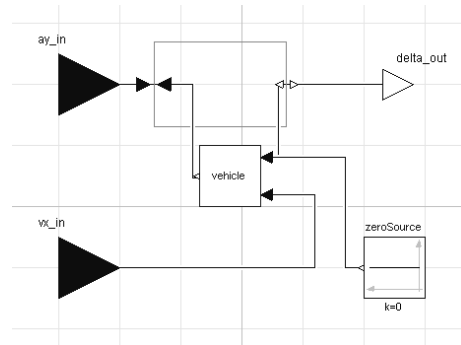


Figure 4: Block diagram of inverse model. Inputs to the model are lateral acceleration and longitudinal velocity at the first unit. Output is the required steering angle at the first axle.

To test the inverted model, a triangular acceleration signal of 0.2 Hz frequency and  $2 \text{ m/s}^2$  amplitude is sent as input. The velocity is 50 km/h. The steering output from the inverted model is then input to a model with normal causality to verify that the lateral acceleration achieved will be correct. Figure 5 shows that the lateral acceleration output from the controlled model is indeed the same as the desired lateral acceleration input. The resulting steering command is shown in figure 6.

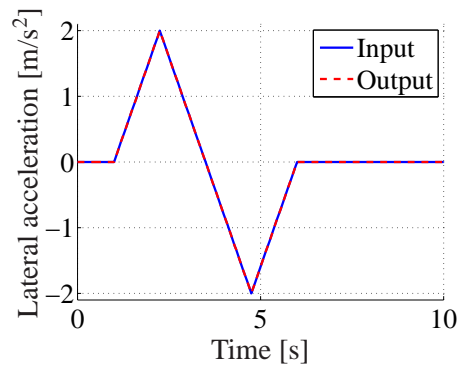


Figure 5: Desired lateral acceleration input and output from vehicle controlled with steering output from inverted model.

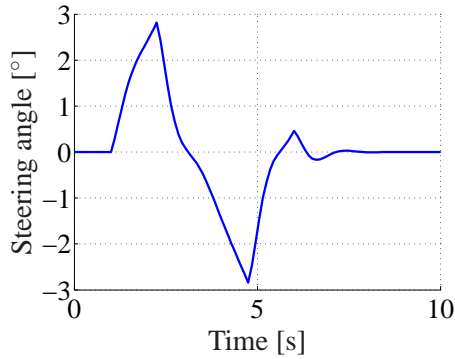


Figure 6: Steering angle from inverted model.

### 3.2 Frequency response

The model can easily be linearized to find the frequency response. An important use case for this is to find how different parameters affect the frequency response. Here, the coupling position between the tractor and the trailer is varied to show how this affects the stability of the vehicle combination.

The truck-dolly-semitrailer combination is linearized for straight forward driving at 80 km/h. Figure 7 shows the baseline gain from steering angle to the yaw rates of the different units in the truck-dolly-semitrailer combination.

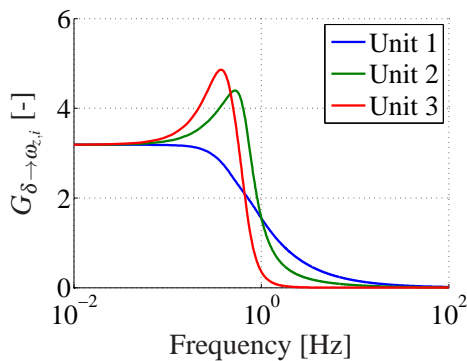


Figure 7: Gain from steering angle at the front axle to the yaw rates of the different units for the baseline parametrization of the truck-dolly-semitrailer combination.

Often the rearward amplification is an important measure of vehicle performance. This is defined as the gain from the first unit yaw rate to each of the towed units' yaw rates. Figure 8 and figure 9 show how the rearward amplification of the dolly and semitrailer changes as the coupling position on the truck is changed. The baseline position is 0.53 m behind the rear axle. As the coupling moves rearward the amplification increases.

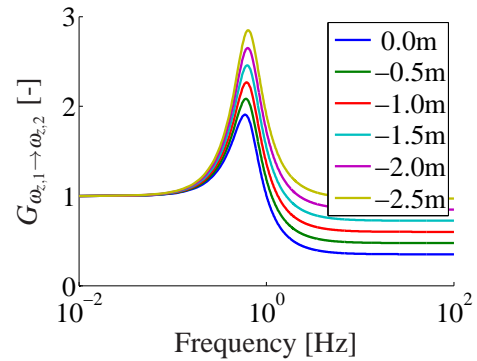


Figure 8: Yaw rate gain from first unit to second unit when moving coupling position on first unit. Coupling further rearward (negative adjustment) gives higher amplification of yaw rate.

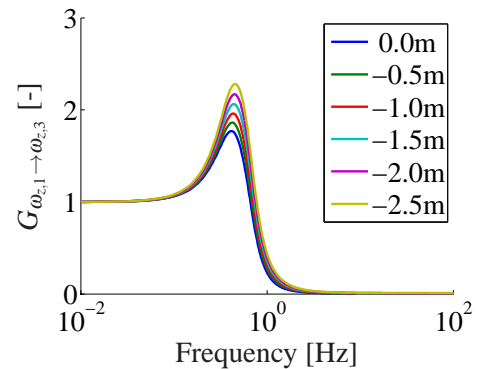


Figure 9: Yaw rate gain from first unit to third unit when moving coupling position on first unit. Coupling further rearward (negative adjustment) gives higher amplification of yaw rate.

### 3.3 Steady-state off-tracking

Off-tracking is an important property of a vehicle with trailers. It is usually defined as the difference in curve radius between the towing unit and the trailers. In the model, the instantaneous curvature is calculated for all axles as

```

for i in 1:nu loop
  for j in 1:na loop
    curvature[i,j] = wz[i]
    /sqrt((vy[i]+Lcog[i,j]*wz[i])^2
    +vx[i]^2);
  end for;
end for;
    
```

To show how off-tracking varies with vehicle speed, the vehicle is driven at a constant curve radius of 100 m for velocities from 1 to 80 km/h. Figure 10 shows how the curve radius of the last axle of each trailer compares to that of the first axle on the tractor. The typical

characteristic is that trailers track cuts into the curve at low speed due to vehicle kinematics and geometry. At high speed the trailers track a larger curve radius due to the increased lateral load causing higher sideslip.

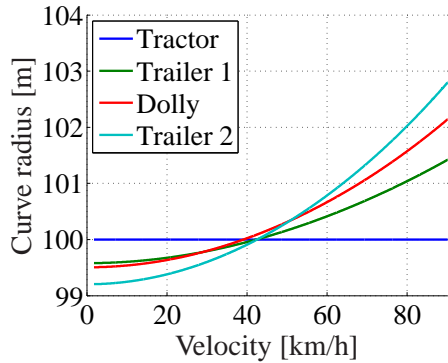


Figure 10: Curve radius of the first tractor axle and the last axle of each towed unit for varying velocity.

Off-tracking for a fixed vehicle speed also varies with the curve radius of the first unit. To study this, the vehicle is run at a fixed speed of 10 km/h and curve radius of the first unit varies from 15 to 100 m. Figure 11 shows how off-tracking varies with the curve radius of the first axle. At this relatively low speed, vehicle kinematics cause more inward off-tracking at smaller curve radii.

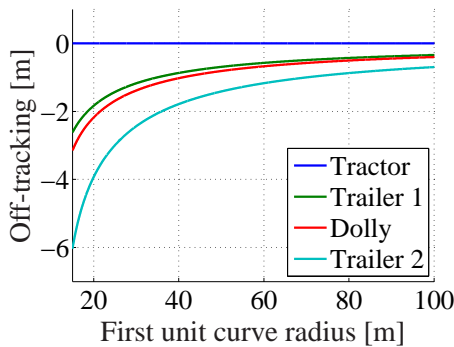


Figure 11: Off-tracking of the last axle of each towed unit in the A-double. First unit speed is fixed and curve radius varies. Positive off-tracking is defined outwards in the curve.

### 3.4 Dynamic simulation

The baggage carrier is simulated with a single period sine steering input of  $5^\circ$  amplitude and 0.3 Hz frequency at 18 km/h. Figure 12 shows the yaw rates of all the units in the airport baggage carrier and figure 13 shows the position of each axle.

The parameters are only approximated and are not measured or estimated from a specific real vehicle.

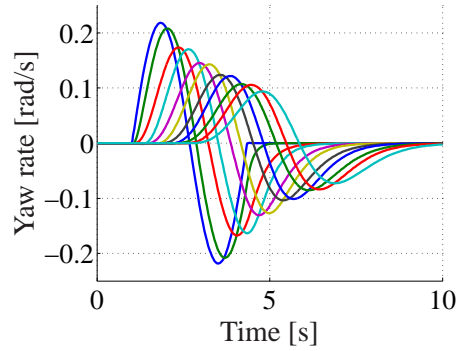


Figure 12: Yaw rates of all the units in the baggage carrier train

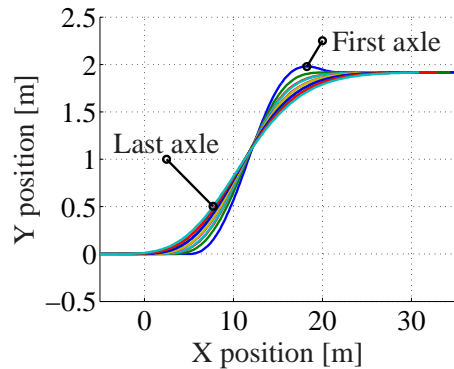


Figure 13: Positions of each axle in the baggage carrier train.)

The simulation results show that the model can represent multi-trailer vehicles without additional modeling effort.

## 4 Conclusion

This work shows that Modelica is useful for formulating simple vehicle models. Using Modelica, articulated vehicle combinations with arbitrary number of units and axles can be modeled with the same level of abstraction as when modeling a non-articulated single-track vehicle model. Tool independent export and import possibilities within the Functional Mockup Interface greatly simplifies the use of models like this in control design and concept studies.

By incorporating simple-but-relevant models together with higher fidelity models in development processes infeasible concepts can be ruled out early and only the most promising ones can be carried on to more detailed analyses.

## References

- [1] M. Levén, A. Sjöblom, M. Lidberg, and B. Schofield, “Derivation of linear single-track truck-dolly-semitrailer model with steerable axles,” Department of Applied Mechanics, Chalmers University of Technology, Tech. Rep. 2011:09, 2011.
- [2] M. Gäfvert and O. Lindgärde, “A 9-dof tractor-semitrailer dynamic handling model for advanced chassis control studies,” Department of Automatic Control, Lund Institute of Technology, Tech. Rep. ISRN LUTFD2/TFRT-7597-SE, 2001.
- [3] S. Kharrazi, “Steering based lateral performance control of long heavy vehicle combinations,” Ph.D. dissertation, Chalmers University of Technology, 2012.
- [4] D. Vazquez-Vega, “Directional analysis of an a-train double with damped articulation,” Master’s thesis, Concordia University, Montreal, 2000.





# Multibody Model of a Motorbike with a Flexible Swingarm

Gianni Ferretti<sup>1</sup> Bruno Scaglioni<sup>2</sup> Andrea Rossi<sup>2</sup>

<sup>1</sup>Politecnico Di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria DEIB  
Via Ponzio 34/5, 20133 Milano, Italy

<sup>2</sup>MUSP Lab, Via Tirotti 9, Le Mose, 29122 Piacenza, Italy

## Abstract

Considering the specific case of the multibody modelling of a racing motorbike, where the rigid model of the rear swingarm has been replaced with a flexible one, a general approach to flexible multibody systems modelling in Modelica is presented in this paper. In particular, the steps required to generate the model of a flexible body starting from a FEM analysis, performed with commercial packages, are detailed. Simulations results are shown with reference to a sudden braking and to a series of impacts with curbs. In this last case, an unstable behaviour occurred when considering the flexible component, which is currently under investigation.

*Keywords: motorcycle dynamics; flexible multibody systems; finite element method; floating frame of reference; Craig-Bampton method.*

## 1 Introduction

Object-oriented modelling, favoring a real modular and multi-domain approach, has been recognized as a fundamental tool for the mechatronic design, requiring an integrated approach to mechanical, electronic and control design [1]. In this respect, even if multibody dynamics is frequently just one of the physical domains involved, the simulation of flexible multibody systems plays an important role.

On the other hand, in real applications, the task of modelling distributed flexibility cannot be addressed without the help of finite elements (FE) codes, in order to describe complex geometries and material properties. Moreover, for the sake of efficiency of numerical simulation, the huge number of nodal coordinates introduced by FE modelling, must be reduced to a much smaller number of modal coordinates, for example through the classical Craig-Bampton method [2].

Two commercial packages exist that pre-process the output of FE codes to get the Modelica model of a

flexible body, one has been developed by the German Aerospace Center (DLR) [3], the other has been developed by Claytex Services Ltd.

The DLR FlexibleBodies library provides several Modelica classes, namely a flexible beam model (`Beam`), an annular plate model (`AnnularPlate`), a thermoelastic plate (`ThermoElasticPlate`) and a model for general flexible bodies exported from FE codes (`ModalBody`). The results of the FE analysis performed by several general purpose codes are first processed by another commercial code: FEMBS, implementing modal reduction in a two-step process. Guyan or Craig-Bampton reduction methods are applied in the first step to keep the flexible body input file to FEMBS small, while in the second step the modes in the frequency range of interest for multibody simulation are selected. The reduced modal representation is then stored in a Standard Input Data (SID) file [4, 5], an object-oriented data structure developed to define a standard format to exchange data between FE and MBS codes. When a `ModalBody` class is instantiated the user has to specify the name of the SID file containing the modal description of the body, which also stores the original mesh of the FE model, used by Dymola to perform the animation of the simulated motion of the flexible body.

The Claytex library generates directly the Modelica model of a flexible body from the output of the model reduction process performed by three FE codes: namely Nastran, Genesis and Abaqus.

Object-oriented modelling of general flexible multibody systems has been also described in [6] (up to the Modelica code), based on the parameters of the flexible body computed by FEM packages, stored in an ASCII file and read by a parsing tool. The model also maintains the efficient choice of the generalized coordinates implemented in the Modelica standard (rigid) multibody library. Thus, when a body is a component of a tree structure, the motion of the local Floating Frame of Reference (FFR) [7] is actually calculated by

propagating of the kinematic quantities from the root of the tree while, in the case of floating bodies, the body itself is a root, introducing its own generalized coordinates for position and orientation.

In this paper, the above mentioned approach is applied to the multibody model of a racing motorbike, where a flexible model of the rear swingarm has been considered.

First, a rigid multibody model of the motorbike has been developed, with the aim of providing forces and torques applied to the frame and to the swingarm for a following FEM structural analysis, performed for the sake of mechanical design validation. As reference simulation scenarios a sudden braking and a series of impacts with obstacles (curbs) have been considered. Then, the rigid model of the rear swingarm has been replaced with a flexible one and the simulation results have been compared.

In view of the particular simulation experiments considered only limited differences were expected, particularly appreciable during the simulation of impacts. On the other hand, the simulation of the motorbike with a flexible swingarm showed an unstable behaviour in the case of subsequent impacts, largely due to a poor performance of the virtual driver, but undoubtedly induced by swingarm flexibility, since the said behaviour in the rigid case did not occur.

The paper is organized as follows. In Section 2 the rigid multibody model of the motorbike is described, including the simulation scenarios and the related results, moreover, the simulation results are compared to experimental results obtained on the real bike. Section 3 explains the flexible multibody modelling approach used in this paper, describing the adopted theoretical formulation and the most relevant characteristics of the Modelica flexible body model. Further, it describes the procedure which leads to obtain flexible body models in Modelica. In Section 4 the flexible model of the rear swingarm is described, the procedure outlined in the previous section is adopted to obtain a flexible model of the body, then, in Section 5 the previously described simulations are repeated and the results are compared with respect to the rigid model. Section 6 concludes the paper.

## 2 Rigid multibody motorbike model

A rigid multibody model of a racing motorbike, built by RobbyMoto Engineering S.r.L., has been first developed (Fig. 1).

Since the considered scenarios were a sudden brak-

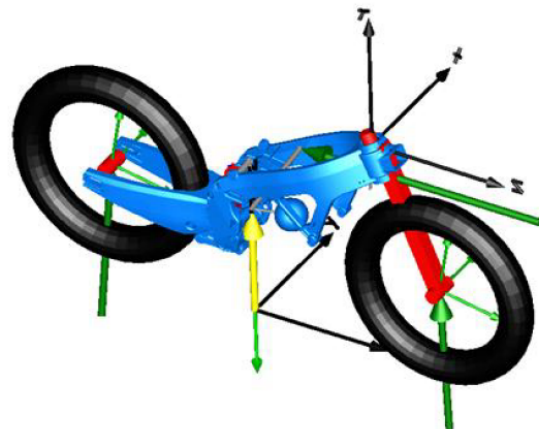


Figure 1: Model of the motorbike.

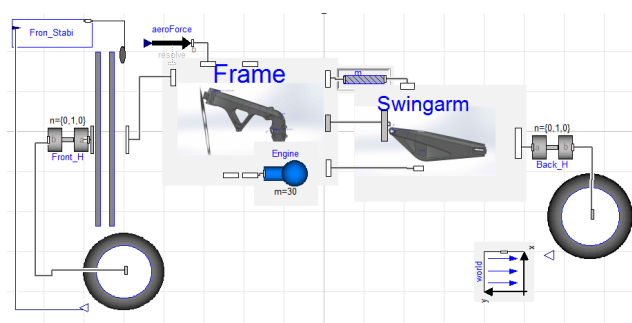


Figure 2: Scheme of the Modelica model of the motorbike.

ing and a series of impacts with obstacles (curbs), the model was not intended to simulate the dynamic behaviour of the motorbike in curves, but a steering degree of freedom was anyway provided, in order to represent the intrinsically unstable behaviour of the vehicle. The adopted tyre model accounted for lateral forces and roll angle, thus a tilt control was required.

The rigid motorbike model is made up entirely by components of the standard Modelica multibody library, except for the wheel/road interaction model and the virtual driver, taken from [8, 9]. The main components are:

- Main frame,
- Front suspensions,
- Rear suspended *Pro-Link* swingarm,
- Wheels and wheel-road interaction,
- Virtual driver: a simple vehicle stabilizer, acting on the front steer and controlling the tilt angle.

and the scheme of the Modelica model is shown in Fig. 2.

It must be pointed out that the suspension mechanism includes two interconnected planar kinematic loops, which must be carefully modelled in order to avoid singularities. In this respect, the Modelica multi-body library provides some aggregate joints implementing an analytic solution of the loops closure equations [10]. In our case, a Revolute-Revolute-Revolute (RRR) joint is used to solve one of the loops, as shown in Fig.4, which also inherently breaks the second one, hence there is no need to use another aggregate joint for the second loop. Figure 3 shows a simplified scheme of the planar loop and how it is realised in the real motorbike.

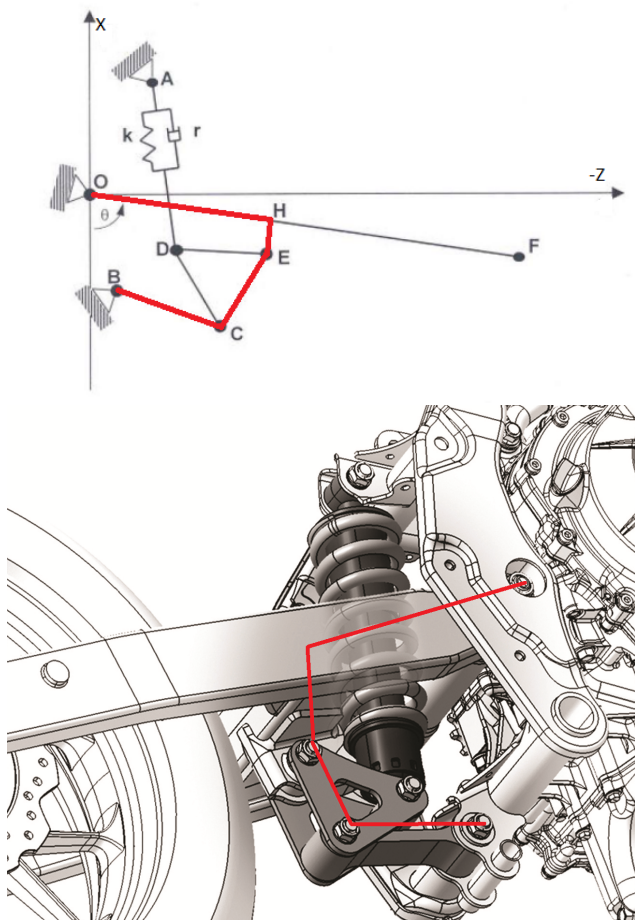


Figure 3: Planar loop.

In view of the simplicity of the considered simulation scenarios, a linear model of the tyre/road interaction has been adopted [8]. The longitudinal force is thus computed as a linear function of the longitudinal wheel slip, and the lateral force is computed as a linear function of the tyre sideslip angle and of the roll angle.

Aerodynamic drag forces have been also taken into account, calculated as:

$$F_a = S_f C_z V^2$$

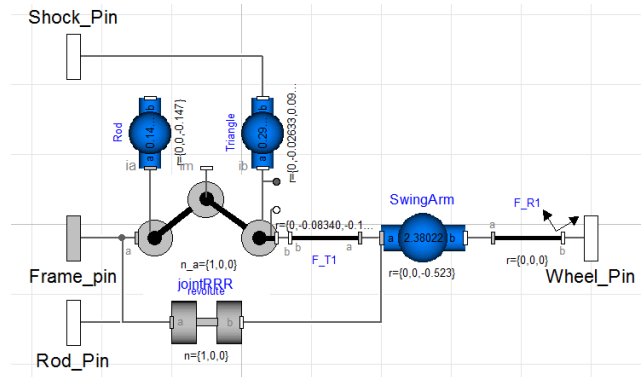


Figure 4: Model of the kinematic planar loop.

Where  $F_a$  is the aerodynamic force applied in the pressure center of the bike, placed 20 cm above the center of mass and biased 10 cm on the front, according to [11]. The front section of the bike is defined as  $S_f$ , the aerodynamic coefficient  $C_z$  is calculated from experimental results ( $C_z = 0.45$ ), and  $V$  is the ground speed of the vehicle. The drag force is shown in Fig. 1 as a green horizontal arrow applied to the pressure center.

### 2.1 Simulation of a braking

In this subsection, the results obtained by simulating a braking are compared to experimental data.

The experiment starts with the bike moving on a straight line at 215 km/h, then, a constant braking torque of 280 Nm is applied to the front wheel for 4 seconds, according to the pressure observed on the front brake of the real bike. While braking, the pilot weight is applied 70% on the front handlebars and 30% on the saddle.

Figure 5 shows two screenshots of the simulation before and during the braking manoeuvre, while Figs. 6 and 7 compare the simulated and experimental speeds and the elongation of the suspensions, respectively. It must be pointed out that the distribution of the driver weight in the Modelica model is constant during the entire simulation, while the distribution of the weight load in the experiment highly depends on the longitudinal acceleration, thus, the front suspension displacement is slightly different between model and experiment before the braking manoeuvre. The comparison between simulation and experiment is more significant in the first part of the transient, say the first 2 seconds, because in the last part the driver enters a curve and starts releasing the brake handle. Anyway, the simulated results appears to be in quite good accordance with the experimental data.

Afterwards, the forces and torques acting on the

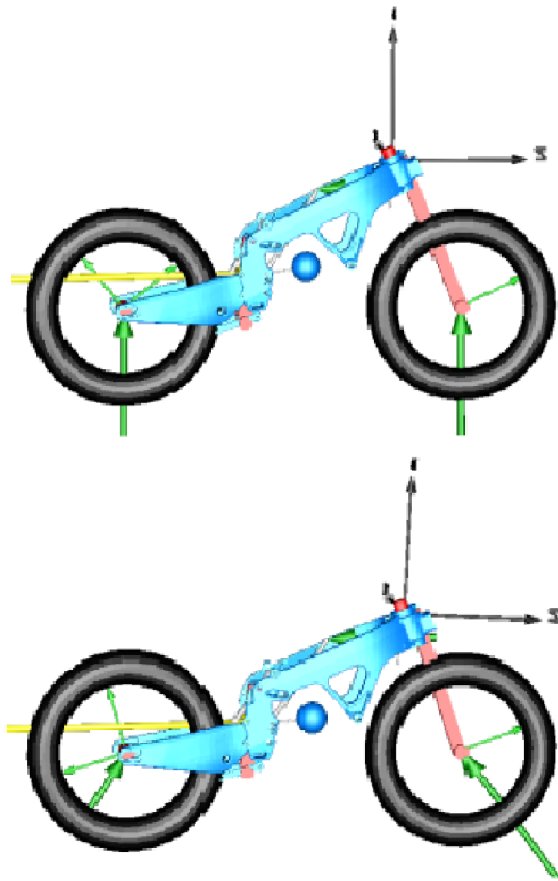


Figure 5: Screenshots of the simulation before and during the braking.

frame and rear swingarm (Fig. 8) have been recorded from the simulation, in order to identify the stress peak and the actual values of loads to be used during a FEM analysis.

### 2.2 Impacts with curbs

In a second simulation experiment a series of impacts with curbs has been considered.

Curbs have been modeled as sawtooth obstacles (Fig. 9) placed on the road surface, every sawtooth has an height of 2 cm and a width of 20 cm, in order to reproduce the real curbs of most racetracks. To this aim, the road model described in [8] has been modified in order compute the quote of the road, given the position of the wheel. Since the driver weight distribution was impossible to estimate in this experiment, the whole load (70 kg) was placed on the saddle. The forces and torques acting on the frame and rear swingarm have been exported for a following FEM analysis step, Fig. 10 shows vertical forces exchanged between rear arm and main frame in the hinge.

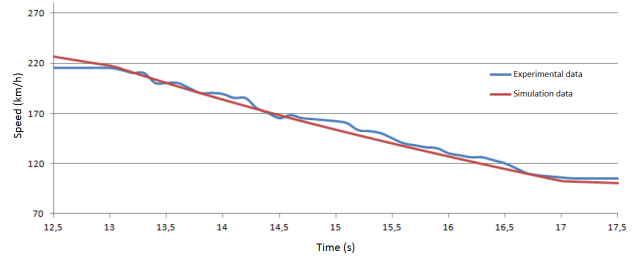


Figure 6: Speeds during braking.

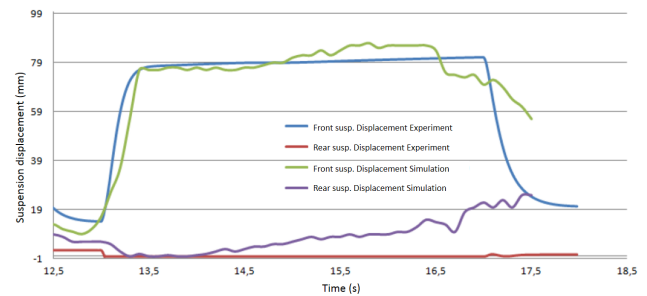


Figure 7: Front and rear suspension elongation during braking.

## 3 Flexible multibody modelling in Modelica

The object-oriented modelling paradigm implemented by the Modelica language requires a description of the dynamics of a flexible body in terms of *local* variables, while the interaction between different bodies has to be described using the *connectors* of the standard Modelica multibody library [10]. In turn, a local description of a body’s dynamics naturally calls for a floating frame of reference (FFR) approach [7], which is currently the most widely used method in computer simulation of flexible multibody systems.

In the FFR formulation, each body is attached to a moving frame of reference undergoing large (rigid) motion, while the (small) elastic displacements are obtained in local coordinates with respect to the reference frame. Thus, the position (in local coordinates) of a point on a flexible body, see Fig. 11, is given by:

$$\bar{\mathbf{u}} = \bar{\mathbf{u}}_0 + \bar{\mathbf{u}}_f, \tag{1}$$

where  $\bar{\mathbf{u}}_0$  is the “undeformed” (i.e., rigid) position vector and  $\bar{\mathbf{u}}_f$  is the deformation contribution to position (i.e., the deformation field).

If small elastic deflections are considered, according to the classical Rayleigh-Ritz method [12], the infinite dimensional deformation field on the body can

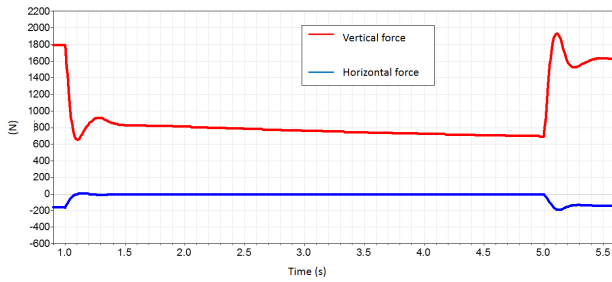


Figure 8: Forces acting on the rear swingarm hinge during braking.

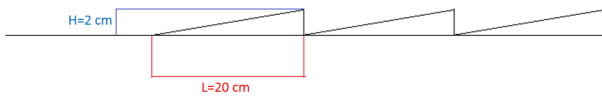


Figure 9: Curbs model.

be approximated by a functional basis space with finite dimension, say  $M$ , so that the vector  $\bar{\mathbf{u}}_f$  can be expressed by the finite dimensional product

$$\bar{\mathbf{u}}_f = \mathbf{S}\mathbf{q}, \quad (2)$$

where  $\mathbf{S}$  is the  $[3 \times M]$  shape functions matrix (i.e., a matrix of functions defined over the body domain and used as a basis to describe the deformation field of the body itself) and  $\mathbf{q}$  is the  $M$ -dimensional vector of deformation degrees of freedom, or *modal* coordinates. The representation of a generic flexible body in the world reference frame requires then  $6 + M$  d.o.f.: 3 corresponding to the rigid displacements  $\mathbf{r}$ , 3 to the undeformed body orientation angles  $\theta$  and  $M$  to the modal coordinates  $\mathbf{q}$ .

Starting from eqs. (1,2) and accounting for the elastic properties of the material and for the mass distribution, the generalized Newton-Euler equations for a generic unconstrained flexible body, formulated with respect to the FFR, can be derived in [13, 7, 14, 15, 5, 16], and developed up to the Modelica code in [6]. It must be also pointed out that the efficient choice of the generalized coordinates, implemented in the Modelica standard (rigid) multibody library, can be maintained. Thus, when a body is a component of a tree structure, the motion of the FFR is actually calculated by propagation of the kinematic quantities from the root of the tree while, in the case of floating bodies, the body itself is a root, introducing its own generalized coordinates for position and orientation.

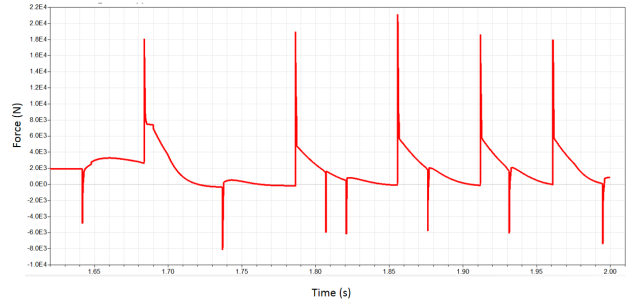


Figure 10: Vertical forces acting on the rear swingarm hinge during impacts with curbs.

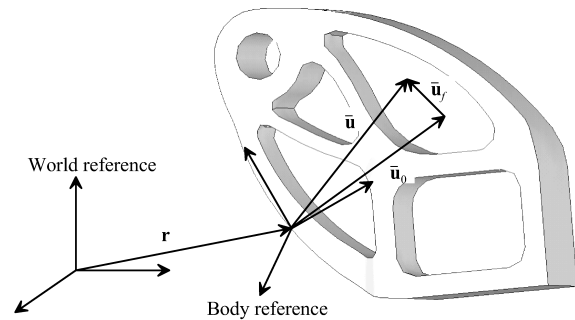


Figure 11: Floating reference frame.

In the case of simple geometries, such as beams [13], the set of data required to implement the flexible body dynamic equations, summarized in Table 1, can be determined analytically, but in more general cases the use of finite elements (FE) computer codes as preprocessors is necessary. In this last case the huge number of *nodal* coordinates must be reduced to a much smaller number of modal coordinates, through the classical Craig-Bampton method [2] or other recently proposed methods [17, 18, 14, 19].

The Modelica model of a general flexible body: FEMBody, is characterized by an array of  $N_c$  multibody connectors, while the data in Table 1 have been suitably collected in the Modelica record BodyData. The record is defined as replaceable:

```
replaceable parameter FEMData.BodyData
data;
```

so that it is possible, by exploiting the features of the Modelica language, to assign a different data record to each FEMBody instance, by simply replacing the record in the model declaration:

```
FEMBody FlexPendulum(redeclare
FEMData.PendulumData data,
alpha=0.005,
```

Table 1: Flexible body data.

$M$	Number of deformation d.o.f.
$I^1, I^2, I_i^3, I^4, I_i^5, I^6, I^7, I_i^8, I_{ij}^9, I_{ij}^{10}, I_{ij}^{11}$	Inertia invariants
$D_e, K_e$	Structural damping and stiffness matrix
$N_c$	Number of connectors
$S_i, \hat{S}_i$	Slices of the modal matrix of connectors d.o.f.
$\bar{u}_{0i}, \bar{A}_i$	Undeformed position and orientation of connectors

beta=0.005,  
d=1);

where alpha, beta, d are the parameters defining the damping matrix (Rayleigh coefficients).

The process of generation of the flexible body data record is schematized in Fig. 12.

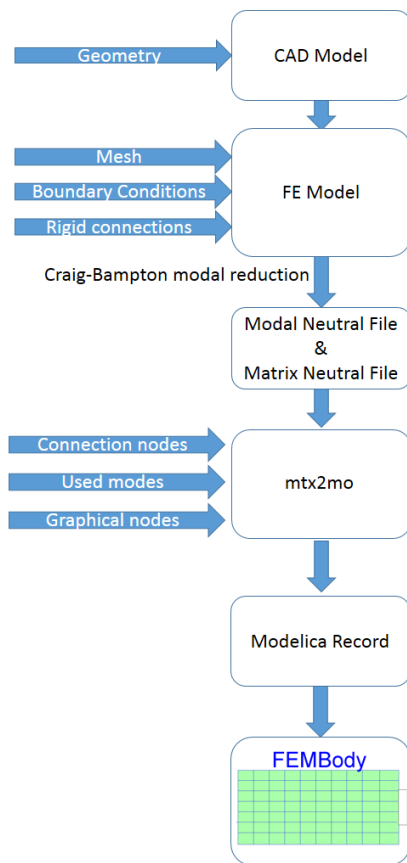


Figure 12: Flexible body data generation.

First, a FE model of the body is developed based on 3D CAD model (often inherited from design phase).

Then, a FEM analysis is performed, essentially consisting in an eigenfrequency analysis followed by a modal reduction step, generally based on the Craig-Bampton method.

The results of the FEM analysis are stored in a bi-

nary Modal Neutral File (.mnf)<sup>1</sup>, which must be then translated into an ASCII file, usually with extension .mtx, containing the same data in a readable format. This step can be performed through the Adams/Flex tool, a package included in the MSC.Adams suite, which allows to inspect the .mnf file and export the content in ASCII format.

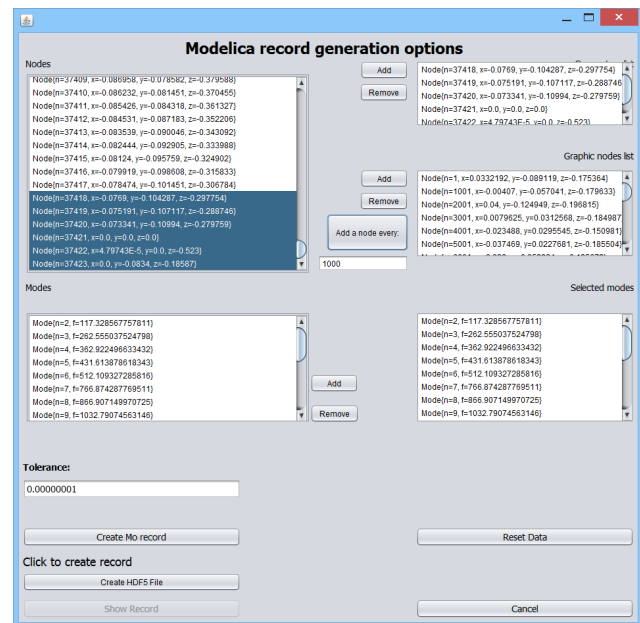


Figure 13: mtx2mo: Graphical User Interface.

The file containing the Modelica record of flexible body data is finally generated by a parsing software tool, named mt<sub>x</sub>2mo, written in Java<sup>2</sup> (a screenshot of the tool is reported in Fig. 13). This tool reads the content of the .mtx file and translates it into the Modelica syntax, moreover, it allows the user to choose:

- the eigenmodes to be considered;
- which nodes of FE processing are selected as the position of the multibody connectors;

<sup>1</sup>The inertia invariants  $I^{10}$  and  $I^{11}$ , are not a direct result of the analysis and are not stored in this file, they are computed by two Modelica functions from invariants  $I^8$  and  $I^9$ , as detailed in [6].

<sup>2</sup>The tool could be implemented also in C or Modelica.

- how many FE nodes are selected for the 3D graphic rendering of the model.

The described approach thus avoids the preprocessing stage adopted by the DLR FlexibleBodies library, which requires the models to be processed by FEMBS in order to obtain the SID file.

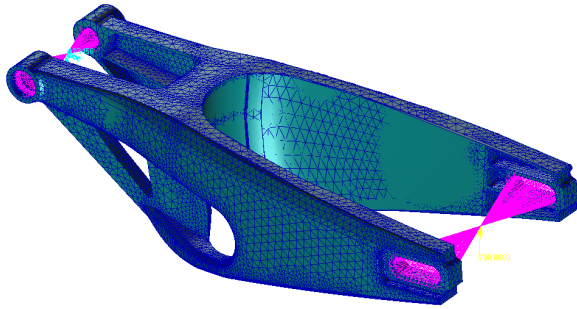


Figure 14: FE model of the swingarm.

#### 4 Modelling a flexible swingarm

Figure 14 shows the FE model of the swingarm, realised with the Patran/Nastran suite by MSC.Software.

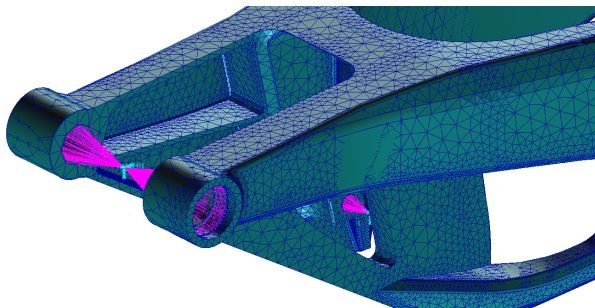


Figure 15: Multi-point constraint.

The swingarm is characterized by three connection points, so three virtual nodes have been defined in the FE geometry: one in the center of the frame pivot, one in the lower triangle pivot, and one in the center of the wheel hub. The internal surfaces of bores have been associated to this massless nodes by RBE2 rigid multi-point connections, namely, every node of the mesh located on the surface of the holes is rigidly connected to the virtual node, so that the motion of all dependent nodes is constrained by the motion of one node. Figure 15 shows the rigid connection between nodes in the front bore.

The boundary conditions were assigned in order to reproduce the hinge acting on the rear swingarm: all translations and two rotations were fixed for the pin

connecting the swingarm to the mainframe, while all the other nodes were free to move. A free-motion eigenvalue resulted from the FEM analysis, with a very small absolute value ( $1.2 \cdot 10^{-3}$  Hz), neglected in the generation of the Modelica model.

The first 20 eigenmodes were retained from the FEM analysis, with eigenfrequencies ranging from 117.3 Hz to 3585.6 Hz, the first eigenmode is shown in Fig. 16.

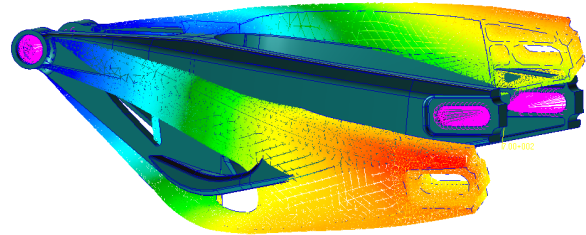


Figure 16: First torsional eigenmode.

It must be pointed out that structural damping is accounted in the flexible body model by means of the Rayleigh coefficients, which are difficult to estimate and are often the result of an averaging on the damping ratios of different modes. In this work the coefficients are chosen as  $d = 1, \alpha = \beta = 0.005$ .

The Modelica model of the flexible swingarm is shown in Fig. 17. Note that, for the sake of modularity, the connectors of the flexible body are stored in a vector, hence it is not possible to distinguish the connections in the graphical layer of the model.

It must be also pointed out that the RRR joint used in the rigid case to manage the kinematic loop is no longer required, as the flexibility of the body inherently breaks the loop.

A relative position sensor has been also introduced, with the aim of sensing the deflection of the rear wheel hub with respect to the position of the rear wheel bore in the undeformed configuration.

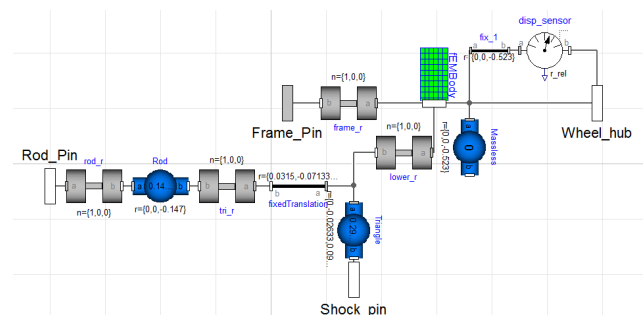


Figure 17: Modelica model of flexible swingarm.

## 5 Simulation results with a flexible swingarm

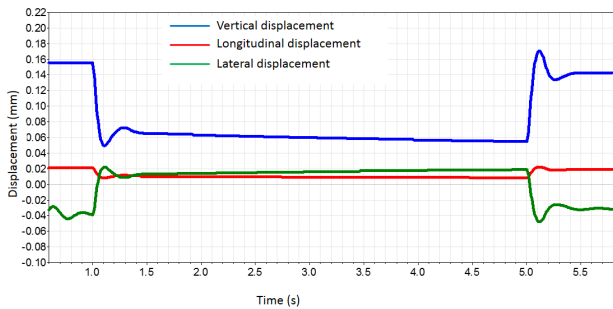


Figure 18: Displacement of rear wheel hub in braking.

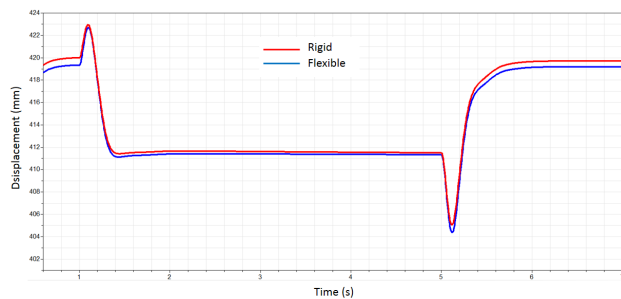


Figure 19: Vertical position of vehicle mass center during braking.

The experiments reported in Section 2 have been repeated with the flexible swingarm model.

Figure 18 shows the deflection of the rear hub measured by the above mentioned relative sensor. During the braking manoeuvre the swingarm deflects under the load on the rear part of the bike, starting from a value of 0.16 mm on the vertical axis before braking. A lateral displacement is also measured, due to the imprecise virtual driver, which cannot maintain the motorbike perfectly vertical. Although the swingarm deflection is small, it anyway affects the overall geometry of the vehicle.

In Fig.19 the quotes of the mass center of the overall vehicle are shown, in the rigid and flexible case. During the braking, the weight and the inertia loads mainly impact on the front wheel, while the rear suspension reaches an equilibrium where poor forces are applied.

Forces and torques applied on the frame do not change significantly with respect to the rigid model, in Fig. 20 the vertical forces in the rear hinge are compared between the rigid and the flexible case at the beginning of the manoeuvre. As expected, the dif-

ferences ( $\approx 3\%$  of the maximum value) are scarcely appreciable in the transients, and the deformation values are in good accordance with the static FE analysis (Fig. 21), in which the same loads (extracted from the multibody analysis) are applied.

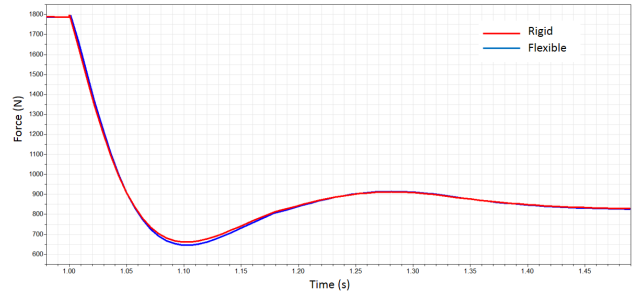


Figure 20: Vertical forces at the beginning of braking manoeuvre.

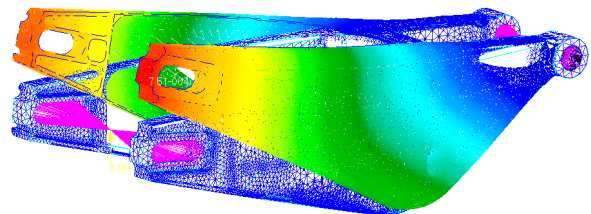


Figure 21: FEM static analysis of rear arm displacement.

During the simulated impacts with curbs the forces acting on the wheel, and consequently on the rear arm, are much higher in magnitude. Figure 22 shows the deflection of the rear wheel hub when the motorbike faces curbs. Note that in this case the vertical deflection reaches values up to 0.6 mm.

Figure 23 shows a comparison between the vertical forces in the rear hinge in the rigid and flexible case. As expected, the forces in the flexible case are lower in absolute value, because part of the energy is used to deform the flexible component, which shows also a dissipative behaviour due to damping.

The overall behaviour of the motorbike in this case changes significantly due to flexibility, Fig.24 shows a comparison of the mass center quote of the motorbike in rigid and flexible case, the differences reach an absolute value of 10 mm. Moreover, when considering the flexible swingarm, an interesting behaviour appears.

If the simulation time is long enough, the virtual driver is no more able to control the vehicle, which starts to wave after some seconds. Figure 25 shows



the motorbike pose in both cases (flexible on the left and rigid on the right) at time  $t = 5.5$  s: the vehicle with a flexible swingarm comes into an unstable behaviour and is going to thumble. This behaviour, currently under investigation, is certainly due to a poor performance of the virtual driver and to a rough model of the tyres, but appears to be induced by swingarm flexibility.

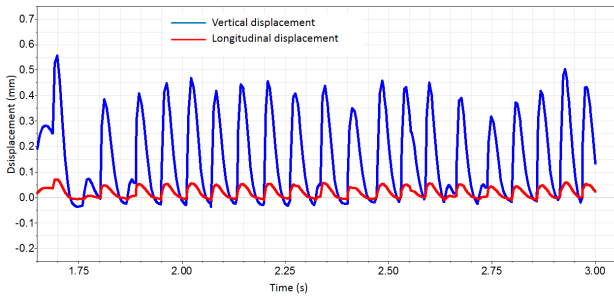


Figure 22: Vertical and longitudinal displacement of rear hub when facing curbs.

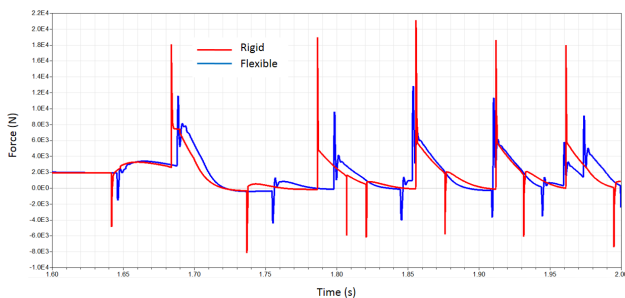


Figure 23: Vertical forces in rear hinge, comparison between rigid and flexible case.

The main drawback of model with the flexible component is the computational cost due to the additional elastic degrees of freedom: the braking simulation experiment with a rigid swingarm takes 0.32 s of CPU time to simulate 7 s, on a normal laptop, the same model with a flexible swingarm takes about 10.4 seconds of CPU time; regarding the simulation of impacts, the rigid model takes 5 s while the flexible model takes 64 s for 3.5 s of simulated time. The DASSL integration algorithm has been used in all the simulations.

## 6 Conclusion and future work

In this paper, a Modelica multibody model of a motorcycle with a flexible swingarm is presented.

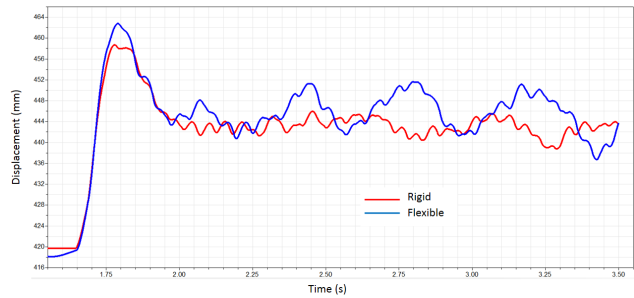


Figure 24: Mass center quote, comparison between rigid and flexible case.

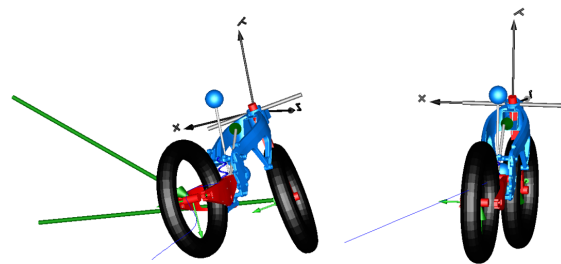


Figure 25: Motorbikes with rigid (right) and flexible (left) swingarm at time  $t = 5.5$  s.

At first, a rigid model of the motorbike has been developed and validated with respect to a sudden braking transient.

Then, a general approach to the modelling of flexible bodies is presented, and the full procedure leading to the Modelica model is detailed.

The proposed modelling approach has been applied to the rear swingarm of the motorbike and a comparison between the rigid and the flexible case is presented, with reference to a sudden braking and a series of impacts with curbs as simulation scenarios. In particular, the simulation of the motorbike with a flexible swingarm showed an unstable behaviour in the case of subsequent impacts, largely due to a poor performance of the virtual driver, but undoubtedly induced by swingarm flexibility.

The developed approach to flexible multibody modelling will allow to easily include the description of bodies' flexibility in mechatronic systems, expanding the range of the dynamic analysis. In particular, the said unstable behaviour is currently under investigation, as well as another unstable behaviour (shimmy) occurring in racing bikes.

## 7 Acknowledgements

The authors would like to thank all people who supported this work: Dr. Marta Massera, for designing the frame and the swingarm, the Director of MUSP, prof. Michele Monno, and its staff for constant support, and RobbyMoto Engineering S.r.L., for valuable collaboration.

## References

- [1] G. Ferretti, G. Magnani, P. Rocco, Virtual prototyping of mechatronic systems, *IFAC Journal Annual Reviews in Control* 28 (2) (2004) 193–206.
- [2] R. R. Craig, M. C. C. Bampton, Coupling of substructures for dynamic analyses, *AIAA Journal* 6 (7) (1968) 1313–1319.
- [3] A. Heckmann, M. Otter, S. Dietz, J. D. López, The DLR FlexibleBodies library to model large motions of beams and of exible bodies exported from nite element programs, in: *5<sup>th</sup> Modelica Conference*, Vienna, Austria, 2006, pp. 85–95.
- [4] O. Wallrapp, Standardization of flexible body modeling in multibody system codes, Part I: Definition of Standard Input Data, *Mechanics Based Design of Structures and Machines* 22 (3) (1994) 283 – 304.
- [5] R. Schwertassek, O. Wallrapp, A. A. Shabana, Flexible multibody simulation and choice of shape functions, *Nonlinear Dynamics* 20 (1999) 361–380.
- [6] G. Ferretti, A. Leva, B. Scaglioni, Object-oriented modelling of general flexible multibody systems, *Mathematical and Computer Modelling of Dynamical Systems* 20 (1) (2014) 1–22.
- [7] A. A. Shabana, *Dynamics of Multibody Systems*, Cambridge University Press, 1998.
- [8] F. Donida, G. Ferretti, S. M. Savaresi, M. Tanelli, Object-oriented modelling and simulation of a motorcycle, *Mathematical and Computer Modelling of Dynamical Systems* 14 (2) (2008) 79–100.
- [9] Marescotti, L., *Modellazione dei sistema pilota-veicolo a due ruote in ambiente integrato Matlab-Adams* (in Italian), Master's thesis, Università½ di Pisa (2003).
- [10] M. Otter, H. Elmqvist, S. Mattsson, The new Modelica multibody library, in: *3<sup>rd</sup> Modelica Conference*, Linköping, Sweden, 2003.
- [11] G. Cocco, *Dinamica e tecnica della motocicletta*, 2013.
- [12] W. Ritz, Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik, *Journal für die Reine und Angewandte Mathematik* 135 (1909) 1–61.
- [13] F. Schiavo, L. Viganò, G. Ferretti, Object-oriented modelling of flexible beams, *Multibody System Dynamics* 15 (3) (2006) 263 – 286.
- [14] J. Fehr, P. Eberhard, Simulation process of flexible multibody systems with non-modal model order reduction techniques, *Multibody System Dynamics* 25 (2011) 313–334.
- [15] U. Lugrís, M. A. Naya, A. Luaces, J. Cuadrado, Efficient calculation of the inertia terms in floating frame of reference formulations for flexible multibody dynamics, *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 223 (2) (2009) 147–157.
- [16] R. Schwertassek, O. Wallrapp, *Dynamik flexibler Mehrkörpersysteme*, Vieweg, Wiesbaden, 1999.
- [17] P. Koutsovasilis, M. Beitelshmidt, Comparison of model reduction techniques for large mechanical systems, *Multibody System Dynamics* 20 (2) (2008) 111–128.
- [18] M. Lehner, P. Eberhard, A two-step approach for model reduction in flexible multibody dynamics, *Multibody System Dynamics* 17 (2007) 157–176.
- [19] C. Nowakowski, J. Fehr, M. Fischer, P. Eberhard, Model order reduction in elastic multibody systems using the floating frame of reference formulation, in: *7<sup>th</sup> Vienna International Conference on Mathematical Modelling - MATHMOD 2012*, Vienna, Austria, 2012.

# Modelling and parameter identification of a semi-active vehicle damper

Michael Fleps-Dezasse   Jakub Tobolář   Johannes Pitzer

German Aerospace Center (DLR)  
Institute of System Dynamics and Control  
82234 Wessling

Michael.Fleps-Dezasse@dlr.de   Jakub.Tobolar@dlr.de   Johannes.Pitzer@dlr.de

## Abstract

In this paper two semi-physical models of the semi-active dampers of the DLR robotic electric vehicle ROboMObil (ROMO) are described and their implementation in Modelica is presented. Besides the damper characteristics and hysteresis, the models additionally consider the gas force and cover the differences of the damper characteristics for compression and rebound. A procedure to identify the damper model parameters was implemented using the DLR Optimization library. The measurement data used for parameter identification was recorded during experiments on a damper test bench. The simulation results of the damper models are compared to the experiment data of the semi-active damper and the suitability of the damper models with respect to accuracy and real-time simulation is discussed.

*Keywords: semi-active damper; model identification, Bouc-Wen model, vehicle dynamics*

## 1 Introduction

When designing a suspension system there are generally two conflicting goals concerning satisfactory ride comfort and good road-holding. Using passive dampers this leads to a compromise since good ride comfort can be achieved by a rather soft damping whereas for good road-holding high damping is necessary. On the contrary, suspension systems with controlled semi-active dampers can mitigate this restriction by allowing the adaption of the damper force according to the current vehicle state. A comprehensive overview of control strategies for semi-active suspensions is given in [1, 2] or [3].

Semi-active dampers enable the continuous adaption of the damper force characteristics within a large

operation range dependent on a control input. The adaption inside the damper is realized by modifying the force generating physical effects of the damper. Therefore, semi-active dampers generally need little energy effort for control as only small forces are necessary to modify the damping force [2]. These two properties, low energy consumption and continuous adaption of the damper force, make the integration of semi-active dampers in vehicle suspensions attractive. In order to investigate the influence of all the aforementioned aspects in the context of an experimental electric vehicle, semi-active dampers are used in the ROMO.

The ROMO, see Figure 1, is an innovative robotic electric vehicle developed at the Robotics and Mechatronics Center of the German Aerospace Center (DLR). It is composed of four *Wheel Robots*, see [4], which integrate drivetrain, brakes, steering, and suspension.



Figure 1: The ROboMObil (ROMO) on the four post test rig

During the design process of semi-active suspen-

sions accurate purpose dependent models describing the behaviour of semi-active dampers are needed as shown e. g. in [5]. These models can then be used for full and quarter-car simulations as well as for estimator and controller design, see [6, 7] or [1].

Due to the ability to combine models from different physical domains like mechanics and/or electrics with control algorithms the object oriented modelling language Modelica is well suited for modelling and simulation of controlled semi-active dampers. Another advantage for modelling of electric vehicles like the ROMO is the possibility to reflect the hierarchical model structure which simplifies the handling and parameterization of large models.

The common component models for semi-active dampers can be separated into the following groups depending on the kind of modelling:

- Full physical models which include a mechanical and an electrical model of the damper as well as a hydraulical model of the fluid flow.
- Semi-physical models which also include the damper mechanics and electrics, but approximate the hydraulics by an empirical model.
- Black-box models – empirical models which do not include physical model information in any way.

A description of a physical damper model can be found in [8, 9] or [10]. A good overview of semi-physical models can be found in [11] or [12]. These two latter papers give an introduction to semi-active damper modelling and describe several semi-physical models like the Bingham model or the Bouc-Wen model. A look-up table based damper model is developed in [13]. The application of a black-box damper model is shown in [14]. There, the authors develop a nonlinear autoregressive exogenous (ARX) model and compare it to an extended Bouc-Wen model according to [11]. The results presented in the paper illustrate that both models reproduce the damper behaviour with a high accuracy while the nonlinear ARX model slightly outperforms the extended Bouc-Wen model.

This paper focuses on two semi-physical damper models – a modified generalized extended Bouc-Wen model and a model based on a 2-dim. look-up table with the damper velocity and the control input as inputs, further referred to as Force Map based damper model. In contrast to the extended Bouc-Wen model presented by [11] and used in [14] and [2], the extended Bouc-Wen model described in this paper dis-

tinguishes between compression and rebound (i. e. decompression) of the damper allowing different damper characteristics. This corresponds to the typical characteristics of semi-active dampers as used in vehicle suspension systems. Further, the dependency of the model parameters on the control input is modelled in a more general way, compared to [11], as the restriction imposed by the linear dependency of the parameters on the control input limits the performance of the extended Bouc-Wen model (see also [14]). Utilizing the implementation in Modelica, the model parameters are identified through a step by step optimization approach using the DLR Optimization library [15]. Hereby, the overall optimization task was split into several smaller subtasks each focusing on a subset of the overall model parameters and making use of particular experiment data specifically recorded for this optimization step. The advantage of this procedure is that the knowledge of the real damper structure and behaviour can be considered in the optimization algorithm and thus local minima finding can be avoided more easily during the optimization. Subsequently, the behaviour of the parameterized damper models is compared to the real damper behaviour and the feasibility of the damper models for real-time simulations is investigated.

The paper is structured as follows. In the next section the semi-active damper used in ROMO is presented. In section 3 the semi-active damper models are introduced and some implementation details are discussed whereas section 4 deals with the experimental setup and the identification approach. Subsequently, section 5 compares the simulated damper behaviour to the real damper behaviour.

## 2 The semi-active damper

In this work, a semi-active dual tube damper from the KW Automotive GmbH with one external controllable electromagnetic valve is used, see Figure 2. The adjustment of the damper force is realized by controlling the electric current flowing through the inductor of the electromagnetic valve. The induced magnetic field determines the position of the valve piston and consequently the oil flow through the valve. The flow direction of the oil through the external valve stays the same during both compression and rebound of the damper as the oil always flows from the rebound volume to the compensational volume.

Besides this electromagnetic valve there are further valves for compression and rebound, similarly to a conventional passive damper. They are placed at the



Figure 2: Semi-active dual tube damper with external continuously variable valve (coil spring was unmounted for experiments)

damper bottom and in the piston thus limiting the oil flow from the compression volume to the compensational volume and from the rebound volume to the compression volume, respectively.

Part of the compensational volume is filled with a pre-stressed gas to compensate the oil volume difference between pulled out damper and compressed damper caused by the piston rod. This gas volume can be regarded as an accumulator that generates a deflection dependent gas force. The gas force increases when compressing the damper and can therefore be considered as a preloaded spring in a damper model.

The electric current to control the damper is generated by a power electric unit which transforms the Pulse-Width Modulated (PWM) signal, further referred to as control input, from the Rapid Control Prototyping (RCP) hardware into the appropriate electric current. Figure 3 illustrates the speed-force map of the damper at the constant control inputs 10% (blue) and 50% (red) as recorded during experiments. The curves represent the raw damper force measurements and therefore reflect both the damper friction and the gas force influences.

The characteristic of the semi-active damper differs for compression (negative velocity) and rebound (positive velocity). This especially can be observed for larger control inputs as significantly higher rebound forces are generated, compared to compression ones, cf. Figure 3.

### 3 Semi-active damper models

The Force Map based damper model and the extended Bouc-Wen damper model were chosen for implementation in Modelica to investigate the achievable accuracy in reproducing the real damper behaviour and to evaluate their real-time feasibility.

In general, they both provide a good approximation of the highly non-linear behaviour of the semi-active

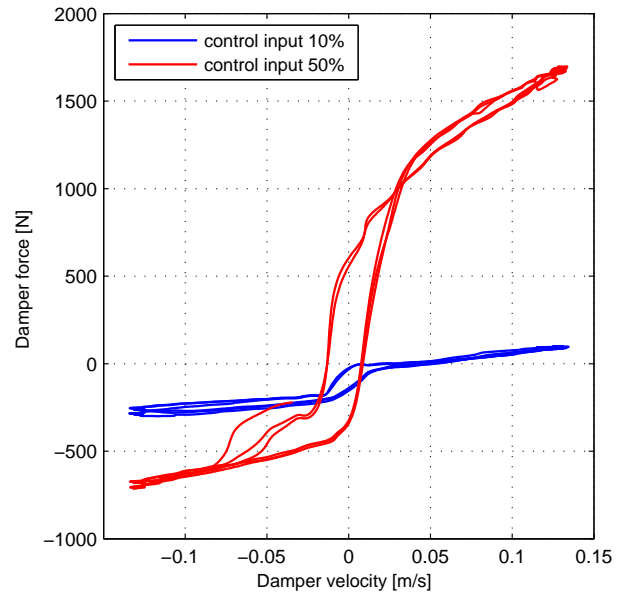


Figure 3: Speed-force map at control input 10% and 50%

dampers, whereas the computational effort is far less than that of physical models as they do not include a model of the damper hydraulics, but approximate it by an empirical model. Therefore, these semi-physical models can be solved more easily by a fixed step solver and are better suited for real-time simulation, state estimation and controller design.

The damper models are implemented as one-dimensional translational models and extend from the common translational interfaces of the Modelica Standard library.

#### 3.1 Generalized extended Bouc-Wen Model

An introduction to the Bouc-Wen hysteresis model can be found in [2]. The extended Bouc-Wen model as shown in Figure 4 is described in detail in [11]. There the authors started with a simpler model consisting of the linear damping element  $c_0$ , the linear spring  $k_0$  and the Bouc-Wen hysteresis model. To better predict the real damper behaviour, they extended this simple model by the elements  $c_1$  and  $k_1$  which reproduce the gas chamber damping and the roll-off at low velocities, respectively. To make the model applicable for semi-active dampers they further determined a subset of three parameters out of the ten parameters of the extended Bouc-Wen model which are then linearly adapted according to the current control input. This way, the extended Bouc-Wen model is able to cover the different characteristics of semi-active dampers de-

pending on the control input. Additionally, the control input was delayed by a first order low pass filter in order to better approximate the delayed real damper response towards changes in the control input due to the dynamic response of the electromagnetic valve and following the dynamic response of the damper hydraulics.

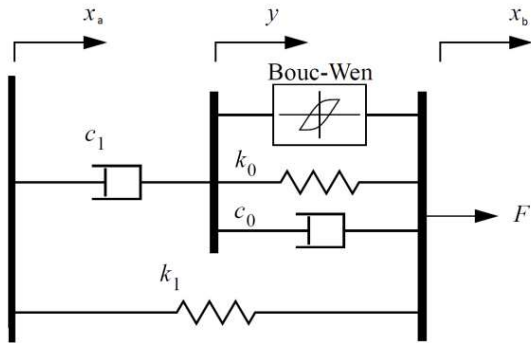


Figure 4: Mechanical model of the extended Bouc-Wen damper model (similar to [11])

Based on the model description in [11], we further generalized the extended Bouc-Wen model as described in the following.

First, the set of three originally suggested parameters depending on the control input is extended to seven parameters and the linear dependency on the control input is replaced by a predefined look-up table, see structure of the implemented Modelica model in Figure 5. This way, the restrictions on the dependency between control input and model parameters are far less than in [11] and the damper model should achieve a better prediction of the real damper. Next, the parameters of the extended Bouc-Wen model are additionally modelled as a function of the current damper velocity to account for the different characteristics of the damper for compression and rebound, cf. Figure 3.

Furthermore, the low pass filter which approximates the dynamic damper response for changes in the control input is modelled as a first order transfer function (block *firstOrder\_varT*) with an additional delay. The time constant as well as the delay time are considered to be functions of the instantaneous damper velocity (block *relSpeed*) and the sign of the derivative of the control input. This way, varying response times  $T_{comp,r}$ ,  $T_{comp,f}$ ,  $T_{rebound,r}$ ,  $T_{rebound,f}$  and delay times  $\tau_{comp,r}$ ,  $\tau_{comp,f}$ ,  $\tau_{rebound,r}$ ,  $\tau_{rebound,f}$  for compression and rebound as well as for rising and falling control inputs, indicated by the indices *r* and *f*, respectively, can be considered.

Utilizing the abovementioned extensions, the result-

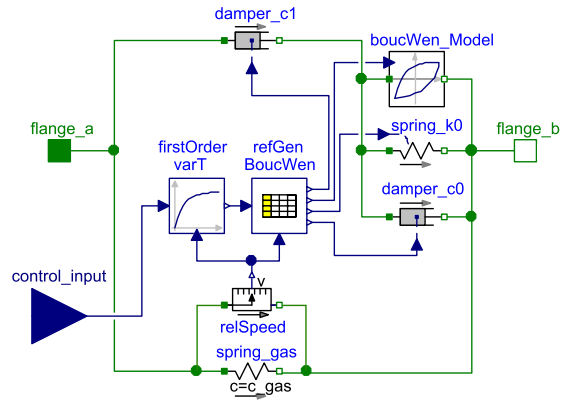


Figure 5: Generalized extended Bouc-Wen damper model

ing equations of the generalized extended Bouc-Wen model are given as

$$F_d = c_0(u, v_d)(\dot{x}_b - \dot{y}) + k_0(u, v_d)(x_b - y) + k_1((x_b - x_a) - x_0) + \alpha(u, v_d)z, \quad (1)$$

with

$$\dot{y} = \frac{1}{c_0(u, v_d) + c_1(u, v_d)} \left( \alpha(u, v_d)z + c_0(u, v_d)\dot{x}_b + c_1(u, v_d)\dot{x}_a + k_0(u, v_d)(x_b - y) \right) \quad (2)$$

and

$$\dot{z} = -\gamma(u, v_d) |\dot{x}_b - \dot{y}| z |z|^{n-1} - \beta(u, v_d)(\dot{x}_b - \dot{y}) |z|^n + \delta(u, v_d)(\dot{x}_b - \dot{y}). \quad (3)$$

The meaning of the displacements  $x_a$ ,  $x_b$ , which correspond to the motion of *flange\_a*, *flange\_b*, respectively, and the internal displacement  $y$  is clear from Figures 4 and 5. The variable  $z$  represents an internal state of the Bouc-Wen model and is often called hysteretic state. The spring stiffness  $k_1$  and the unstretched spring length  $x_0$  model the gas spring of the damper. Therefore, these parameters are considered invariant to changes of the control input  $u$  and the damper velocity  $v_d = \dot{x}_b - \dot{x}_a$ . The remaining parameters of the generalized extended Bouc-Wen model are  $k_0$ ,  $c_0$ ,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$  and  $n$ . All these parameters are functions of the control input  $u$  and the damper velocity  $v_d$  except for the exponent  $n$ , which is empirically determined to be  $n = 2$  as in [2].

It is assumed that the dependency of the parameters on the damper velocity  $v_d$  can be approximated by a switching function which distinguishes between positive damper velocity for rebound and negative one for

compression. This means that for every set  $\theta_{eBW}$  of parameters two variants are generated – one for compression  $\theta_{eBW,comp}$  and one for rebound  $\theta_{eBW,rebound}$ . These sets are then stored in the look-up table mentioned above. To combine the two sets of parameters the following smooth switching algorithm is applied:

$$\theta_{eBW} = s \theta_{eBW,rebound} + (1 - s) \theta_{eBW,comp}, \quad (4)$$

with the switching signal  $s$  which is given by the following sigmoid function:

$$s = 0.5 \tanh\left(\frac{v_d}{v_{eps}}\right) + 0.5. \quad (5)$$

This function generates a smooth output that changes from zero to one in a small region around zero damper velocities  $v_d$ . The size of this region is determined by  $v_{eps}$ . Although other switching signals like  $s = \text{sign}(v_d)$  would be possible, such a smooth switching signal is better suitable for a real-time simulations as no events occur. The function calculating the time constant of the first order transfer function and the delay time from the current damper velocity  $v_d$  and control input derivative is implemented using a similar switching algorithm as the one described in equations (4) and (5).

### 3.2 Force Map based model

Figure 6 shows the Force Map based damper model implemented in Modelica. The control demand applied to the damper model is again (as in section 3.1) modelled by a transfer function and an additional delay with time constants and delay times dependent on the damper velocity  $v_d$  and the control input derivative. The main component of this damper model is a *lookUpTable*, which takes the current damper velocity and the delayed control input as inputs and computes a force. Additionally, a model of the gas spring is included to improve the damper model for large deflections. The overall damper force is therefore given by:

$$F_d = F_{lookup} + F_{gas}, \quad (6)$$

with the output of the look-up table  $F_{lookup}$  and the gas force  $F_{gas}$ .

The implemented model is similar to the one developed in [13]. The main differences between these two models are the negligence of the damper housing stiffness which is very high and therefore causes stability problems during real-time simulations.

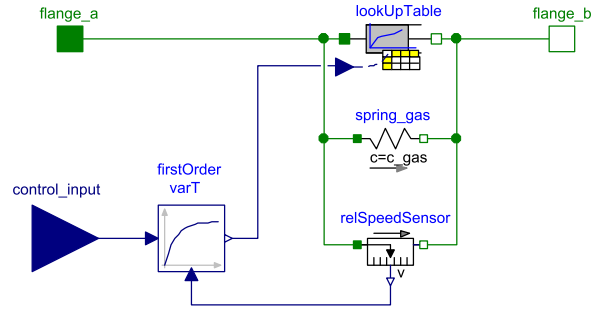


Figure 6: Force Map based damper model

## 4 Parameter identification

The parameter identification of the two semi-active damper models introduced in section 3.1 and 3.2 is a challenging task because both models are highly non-linear and additionally the number of parameters to be determined is large. The arising optimization problem is non-convex and a lot of effort is necessary in order to avoid local minima evaluations. In this work, the optimization problem at hand is split into several subproblems as suggested in [13] and [2], estimating successively subsets  $\theta_j$  of the overall parameter vector  $\theta$ .

The parameter subsets  $\theta_j$  are chosen from experience and technical understanding of the damper in such a way that specific experiments focusing on identification of these parameters can be performed. The suboptimization problems are arranged in a chronological order and the estimated parameter subsets resulting from previous steps are inserted in the current optimization run. Although some of these suboptimization problems are non-convex as well, local minima can be avoided more easily by a careful choice of the initial values of the parameter subsets. This is achievable because the optimization problem is smaller compared to the overall optimization problem.

Throughout all suboptimizations, a cost function

$$J = \frac{\frac{1}{N} \sum_{i=1}^N \left( F_{d,m} - F_{d,s}(\theta_j) \right)^2}{\frac{1}{N} \sum_{i=1}^N \left( F_{d,m} - \left( \frac{1}{T} \sum_{j=1}^N F_{d,m} \right) \right)^2} \quad (7)$$

is used.

This cost function, which is also used by [14], represents the error-to-signal-ratio of the simulated damper force. Here, the number of the measured data samples is denoted by  $N$ . The measured damper force is represented by  $F_{d,m}$  and the simulated damper force by  $F_{d,s}$ .

The suboptimization problems can now be stated by  
 find  $\underset{\theta_j}{\operatorname{argmin}}(J)$  subject to  $\theta_{j,\min} < \theta_j < \theta_{j,\max}$ , (8)

with  $\theta_{j,\max}$  and  $\theta_{j,\min}$  restricting the maximum and minimum parameter values.

The optimization is done in Modelica/Dymola using the model optimization functions of the DLR Optimization library [15]. From the various optimization algorithms available in this library the *Simplex method* was chosen in this work as the algorithm proved satisfactory and fast convergence during preliminary investigations. A default cost function value larger than the initial cost function value was defined in the optimization setup to handle simulation runs with the generalized extended Bouc-Wen damper model which became unstable for certain parameter sets.

#### 4.1 Experimental Setup

The experiments were performed on a Röhrig damper test rig which is shown in Figure 7.

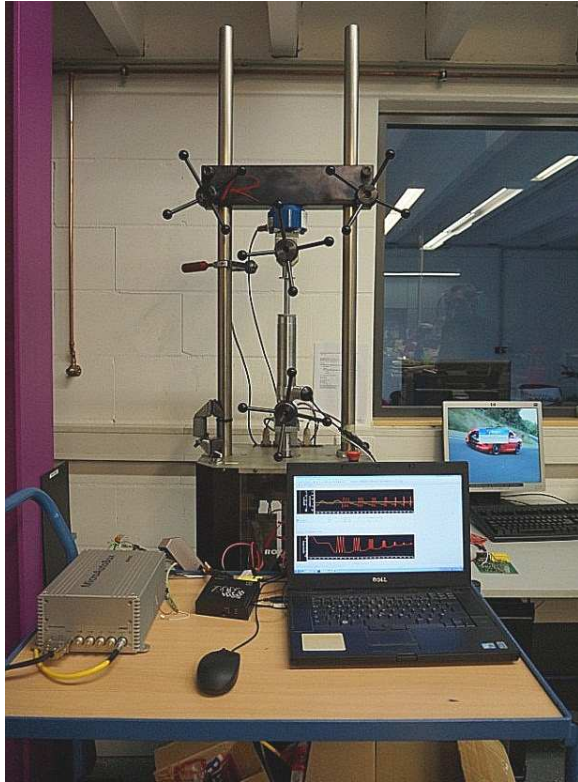


Figure 7: Experimental set-up

This test rig is equipped with a linear electric motor and therefore able to generate various damper excitations like sine sweeps or constant velocity periods. The measurement data was recorded using a Microautobox 2 from dSpace. The sample period of the RCP

system was set to  $T_s = 1$  ms. The measured signals are damper force, displacement and acceleration. These signals are recorded together with the control input and are pre-filtered by a low-pass FIR filter.

#### 4.2 Step by Step optimization approach

The size of the overall parameter vector  $\theta_{eBW}$  of the generalized extended Bouc-Wen model depends on the number of nodes in the look-up table storing the control input dependent parameters as described in section 3.1. Each control input segment is represented by one row in this look-up table. The elements of the parameter vector  $\theta_{eBW}$  are:

$$\theta_{eBW} = (k_0, x_0, \theta_{eBW,i,2}, \dots, \theta_{eBW,i,m+1}, T_{i,r}, T_{i,f}, \tau_{i,r}, \tau_{i,f})^T, \quad (9)$$

with

$$\theta_{eBW,i,j} = (c_{0,i,j}, k_{0,i,j}, c_{1,i,j}, \alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}, \delta_{i,j})^T. \quad (10)$$

Here,  $i$  stays for *comp* or *rebound*,  $j = 2, \dots, m+1$  and  $m$  is equal to the number of control input segments. This parameter vector is split into the following  $m+2$  subsets.

- Gas force identification

$$\theta_{eBW,1} = (k_0, x_0)^T. \quad (11)$$

Experiment: Constant, very small velocity excitation for compression and rebound at constant minimum control input  $u_{\min}$ .

- Bouc-Wen parameter identification at constant control input  $u_1$ :

$$\theta_{eBW,j} = (\theta_{eBW,comp,j}, \theta_{eBW,rebound,j})^T. \quad (12)$$

Experiment: Sine excitation of the damper with an amplitude of 25 mm and a frequency of 0.8 Hz.

- Time constant and delay estimation

$$\theta_{eBW,m+2} = (T_{comp,r}, T_{comp,f}, T_{rebound,r}, T_{rebound,f}, \tau_{comp,r}, \tau_{comp,f}, \tau_{rebound,r}, \tau_{rebound,f})^T. \quad (13)$$

Experiment: Saw tooth excitation of the damper with an amplitude of 25 mm and a constant damper velocity with an absolute value of  $0.15 \text{ ms}^{-1}$  for compression and rebound.



As an example, Figure 8 shows the virtual test rig for the optimization of the Bouc-Wen parameters at control input  $u_1$ . Similar virtual test rigs are employed to optimize the other parameter subsets. Therefore, the measurements have to be replaced by the corresponding measurement data, e. g. with a different control input.

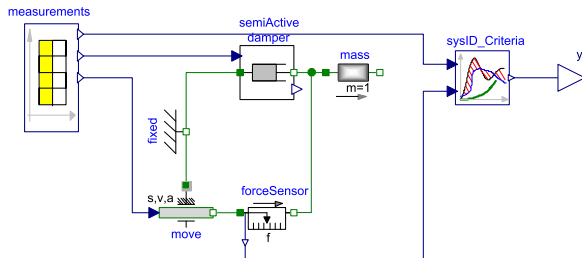


Figure 8: Virtual test rig for optimization

On the left-hand side of the model the measurement data is read by a *CombiTimeTable* called *measurements*, on the right-hand side the cost function is calculated by the *sysID\_Criteria* block and in the middle the damper model *semi-active damper* is excited by a *move* element supplied with the measured displacement and derived velocity and acceleration.

The estimated gas force of step one is inserted as known value in the damper models for the following optimization steps and, further, the estimated parameters  $\theta_{eBW,j}$  (with  $j = 2, \dots, m + 1$ ) of the previous step are used successively as initial values for the parameter vector  $\theta_{eBW,j+1}$  of the following step. The initial values of the extended Bouc-Wen model for the first step are taken from [12] and the corresponding minimum parameter values  $\theta_{j,min}$  and maximum parameter values  $\theta_{j,max}$  are defined by scaling the initial parameter values.

The optimization procedure for the Force Map based damper model is very similar to the one of the extended Bouc-Wen model. The first step gas force estimation and the last step time constant estimation are the same. The steps in between are used to estimate the values of the *lookUpTable*, which takes the damper velocity and the control input as inputs and calculates a force as output. As the control input is constant during these steps, the values are again determined row by row.

### 4.3 Optimization results

The resulting cost function values for both damper models are shown in Table 1. The models reproduce the real damper behavior with almost the same accu-

racy. For a control input of 10% the Force Map model and the extended Bouc-Wen model achieve the same accuracy. For a control input of 50% the extended Bouc-Wen model achieves slightly better values than the Force Map model.

Table 1: Optimal cost function values of the damper models

Damper model	Control input	
	10%	50%
Extended Bouc-Wen	0.097	0.090
Force Map	0.095	0.117

The corresponding force over time, force over displacement and force over velocity diagrams for 10% and 50% control input are illustrated in the figures 9 and 10. The slight accuracy difference indicated by the cost function value for 50% control input is most obvious in the force over velocity diagram (Figure 10 below), mainly in the small velocity region.

In Figures 11 and 12 the results of the time constant and time delay identification for control input steps during rebound and compression are shown. The damper velocity was kept constant for rebound and compression to isolate the damper response for control input steps only. The modelled time behavior of the damper consisting of a first order transfer function and a delay with different time constants and delay times for rebound and compression as well as dependent on the direction of the control input change as described in section 3.1, is able to reproduce the damper behaviour very well. From Figures 11 and 12 it can further be seen that the delay is almost as large as the time constant of the transfer function and therefore the delay cannot be neglected by the damper model.

### 4.4 Real-time suitability

The parametrized damper models were simulated by a fourth-order Runge-Kutta solver with a fixed step size of  $T_s = 1$  ms to analyse their feasibility for real-time simulations, as this solver is used to run the real-time model of the ROMO. The Force Map based damper model showed good results for this solver, while the generalized extended Bouc-Wen model became unstable for the optimized parameters.

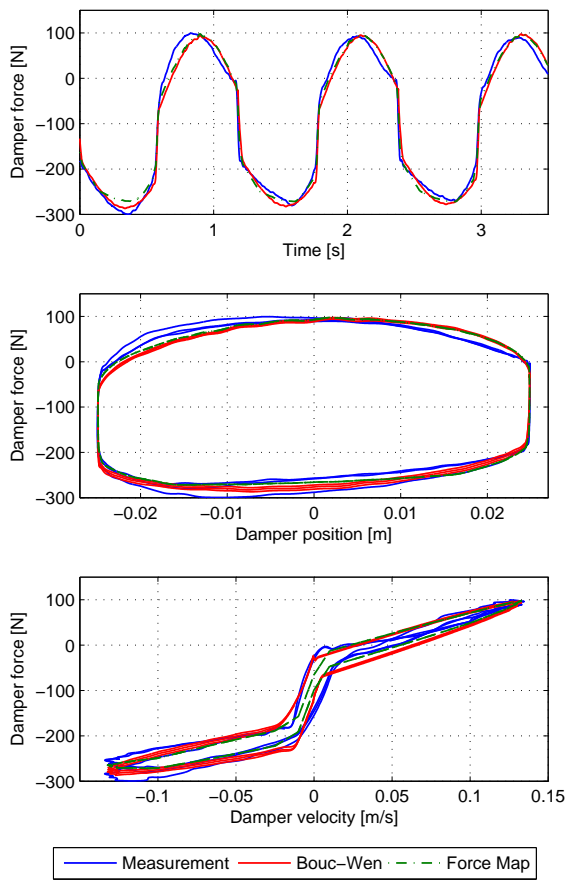


Figure 9: Comparison of damper models to damper measurements at 10% control input (above: force over time; middle: force over position; below: force over velocity)

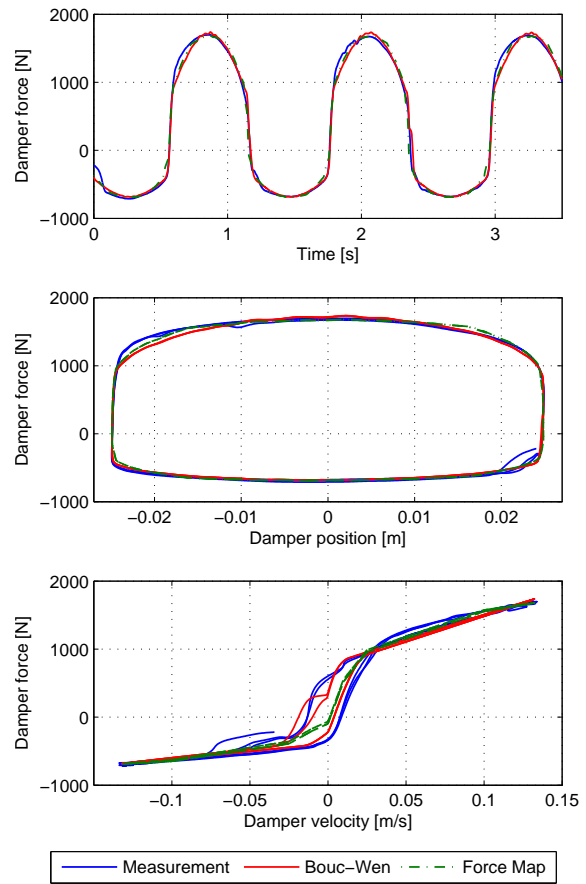


Figure 10: Comparison of damper models to damper measurements at 50% control input (above: force over time; middle: force over position; below: force over velocity)

## 5 Conclusion

In this paper two semi-physical damper models are presented and their accuracy in reproducing the real damper behavior is compared. Both models achieve a high accuracy in the considered velocity and frequency range. The generalized extended Bouc-Wen model gives more accurate results especially for higher control input.

On the other hand, the Force Map based damper model proved a better suitability for real-time simulation due to its more robust stability properties depending on the chosen solver and step-size and its simpler structure.

In the future, the velocity and frequency range of the damper models have to be extended towards higher values to improve the damper models for larger operation ranges. Further, the influence of stiction and sliding friction on the damper model accuracy for small

control inputs and small velocities have to be investigated as well.

## Acknowledgement

The authors would like to thank Dr. Johann Bals and Mr. Jonathan Brembeck for their support and Mr. Uwe Bleck for his expertise in questions of applied vehicle dynamics. A special thank of the authors goes to Dr. Andreas Pfeiffer for his great support in application questions concerning the DLR Optimization library. The authors further would like to thank the KW Automotive GmbH for providing the semi-active damper and the damper test rig.

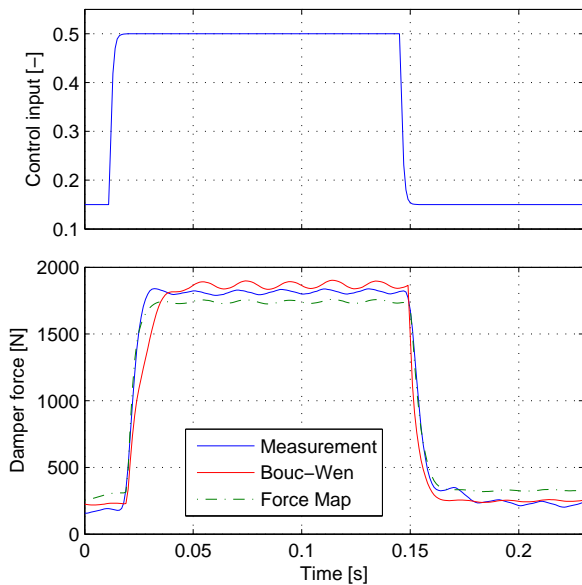


Figure 11: Force over time diagram with control input steps from 15% to 50% and back (during rebound)

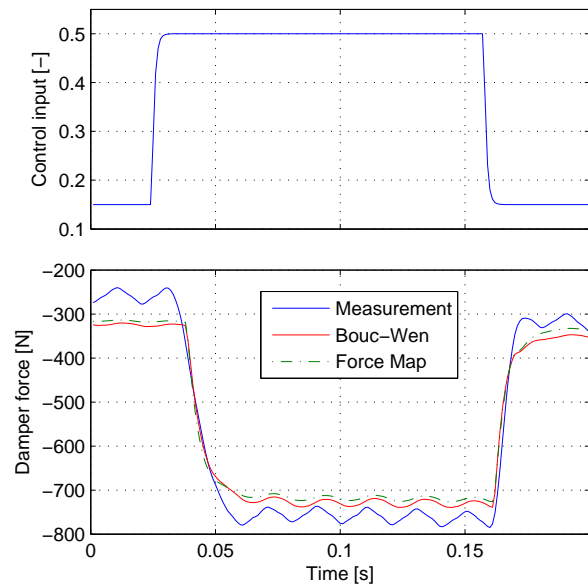


Figure 12: Force over time diagram with control input steps from 15% to 50% and back (during compression)

## References

- [1] Savaresi S. M.: Semi-active suspension control design for vehicles. Amsterdam, Boston: Butterworth-Heinemann/Elsevier, 2010.
- [2] Guglielmino E.: Semi-active suspension control: Improved vehicle ride and road friendliness. London: Springer, 2008.
- [3] Valášek M. and Kortüm W.: EU-Project COPER-NICUS: Development of Semi-Active Road-Friendly Truck Suspension: Choice of Control Law, Experimental Verification, Implementation, Results. Aachen, 2000.
- [4] Brembeck J., Ho L. M., Schaub A., C. Satzger, Tobolar J., Bals J. and Hirzinger G.: ROMO – the robotic electric vehicle. In: International Association for Vehicle System Dynamics (IAVSD), 2011.
- [5] Kortüm W., Valášek M., Šika Z., Schwartz W., Steinbauer P., and Vaculín O.: Semi-active damping in automotive systems: Design-by-simulation. International Journal of Vehicle Design, vol. 28, no. 1, pp. 103–120, 2002.
- [6] Fleps-Dezasse M. and Brembeck J.: Model Based Vertical Dynamics Estimation with Modica and FMI. In: Advances in Automotive Control, Elsevier, IFAC, pp. 341–346, 2013.
- [7] Koch G., Kloiber T. and Lohmann B.: Nonlinear and filter based estimation for vehicle suspension control. In: 49th IEEE Conference on Decision and Control (CDC), pp. 5592–5597, 2010.
- [8] Duym S. W., Stiens R., and Reybroek K.: Evaluation of Shock Absorber Models. Vehicle System Dynamics, vol. 27, no. 2, pp. 109–127, 1997.
- [9] Duym S. W.: Simulation Tools, Modelling and Identification, for an Automotive Shock Absorber in the Context of Vehicle Dynamics. Vehicle System Dynamics, vol. 33, no. 4, pp. 261–285, 2000.
- [10] Pellegrini E., Koch G., and Lohmann B.: Physical Modeling of a Nonlinear Semi-Active Vehicle Damper. In: Advances in Automotive Control: IFAC, Elsevier, pp. 324–329, 2010.
- [11] Spencer Jr. B. F., Dyke S. J., Sain M. K. and Carlson J. D.: Phenomenological Model for Magnetorheological Dampers. J. Eng. Mech, vol. 123, no. 3, pp. 230–238, 1997.
- [12] Butz T. and von Stryk O.: Modelling and Simulation of Electro- and Magnetorheological Fluid Dampers. ZAMM – Journal of Applied Mathematics and Mechanics, vol. 82, no. 1, pp. 320, 2002.

- [13] Altmann F.: Identifizierung eines magnetorheologischen Dämpfers: Vorbereitung des Modells eines leichten individuellen Stadt-Autos (LISA) zur Simulation unter Verwendung des semi-aktiven Dämpfers. Diploma thesis, 2004.
- [14] Savaresi S. M., Bittanti S. and Montiglio M.: Identification of semi-physical and black-box non-linear models: the case of MR-dampers for vehicles control. *Automatica*, vol. 41, no. 1, pp. 113–127, 2005.
- [15] Pfeiffer A.: Optimization library for interactive multi-criteria optimization tasks. In: *Modelica 2012 Conference*, 2012.

# The Modelica *HouseModels* Library: Presentation and Evaluation of a Room Model with the ASHRAE Standard 140

Ana Constantin Rita Streblow Dirk Müller

RWTH Aachen University, Institute for Energy Efficient Buildings and Indoor Climate  
Mathieustr. 10, 52074 Aachen

## Abstract

As part of its contribution to IEA Annex 60, the Institute for Energy Efficient Buildings and Indoor Climate of RWTH Aachen University will make its Modelica *HouseModels* library available. The scope of this paper is to provide information about the library. The first part presents the library and its functionality. In the second part a room model is evaluated by using Case 600 from the test suite provided by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) in the standard 140.

*Keywords:* *HouseModels* library, IEA Annex 60, ANSI/ASHRAE Standard 140, building performance simulation

## 1 Introduction

The Institute for Energy Efficient Buildings and Indoor Climate of RWTH Aachen University aims through its research to reduce the energy consumption of buildings and increase indoor air quality, while focusing on energy generation, storage, distribution and delivery to rooms, buildings or city districts. Over the years several libraries have been developed for the dynamical simulation of building energy systems [1].

On a simulation level, modeling the thermal behavior of the construction body and the building envelope has to mediate the following trade-off: CPU(central processing unit)-time and detail. The building models serve on a basic level as an energy consumer. However through accurate modeling of heat transfer inside the building it is possible to assess the thermal comfort, while taking into account more subtle effects like the storage of energy into the thermal mass of the building. As part of its contribution to IEA Annex 60 [2] the institute will make its Modelica *HouseModels* library available.

The *HouseModels* library aims to provide standard

models for one family dwellings (stand alone house), single apartments and multi-family dwellings consisting of several apartments. The particularity of this library lies in providing ready to use models for the dynamic simulation of building energy systems, while allowing for a degree of flexibility in adapting or extending these models to ones needs.

A library with models for standard houses as such does not yet exist. While at the moment the standard house models are tailor-made for the German market, it is possible to adapt them to other markets.

For the building models we used components from our *Building* library [1]. When setting up detailed house models, we follow an approach of building each room individually and of having each room component visible on the room model level. This approach was formed with time by dealing with project partners in our research activities and with students in our teaching activities. We analyzed how easy it is to understand the model and how quickly one can start working with it. As a consequence the graphical description of a model can quickly get cluttered, so we try to keep our detailed house models as simple as possible. An advantage of simple models with few equations is that, when set up correctly, they can lead to short CPU-times for simulations. In order to make sure that these models are correct they need to be evaluated. We choose the ANSI/ASHRAE Standard 140 [3] for evaluating or models.

Several Modelica libraries for building components already exist, some of them free [4] and [5]. While the *Buildings* library is already validated [6], we chose to use our own components when building the house models in order to maintain consistency between our libraries and because our models are simpler than the ones from the *Buildings* library. We need them as simple as possible when for example doing whole year simulations for a set up consisting of a building and its heating system, especially when the focus lies on the components in each room. For a direct comparison

between our room model and the one from the *Buildings* library please refer to [4] and section 2.2 in this paper, which in order to allow for an easy comparison are similarly structured.

The first part of the paper presents the *HouseModels* library, while the second one presents the first steps in the ongoing work of evaluating our models with the ASHRAE Standard 140: the results for the case 600.

The library was developed using the commercial simulation environment Dymola [7].

## 2 The *HouseModels* library

When developing the *HouseModels* library we followed several goals:

- develop standard models
- model only the necessary physical processes
- build a model so that changing the parameters is easy, quick and will not lead to hidden mistakes
- have an easy to use graphical interface
- ensure a degree of flexibility for expanding or building new models

We call these house models *standard* for the following reasons:

- the floor layouts were made based on existing buildings, by analyzing data provided by the German Federal Statistical Office and by consulting with experts
- for modeling realistical wall structures building catalogues as well as experts were consulted
- the physical properties of the materials for the wall layers were chosen to satisfy the insulation requirements of current and past German energy saving ordinances (e.g. [8])

### 2.1 Library structure

Figure 1 presents the structure of the library. The package *Rooms* contains room models for one (OFD) and multi-family dwellings (MFD).

The multi-family dwelling is based on an existing building consisting of several identical apartments which is part of a larger national research project [9]. The dimensions and layout of the rooms are fixed, with

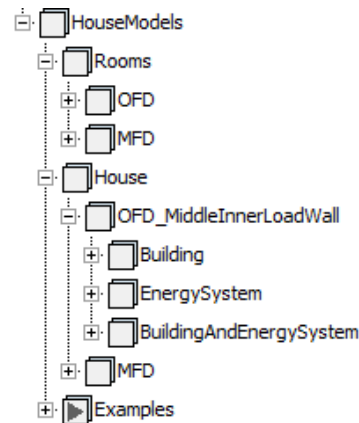


Figure 1: Structure of *HouseModels* library

an apartment having a living area of 70 m<sup>2</sup> and consisting of a living room, two bedrooms, a kitchen and a bathroom.

The one family dwelling isn't based on an existing building, but on a virtual two storey building with ten rooms and a saddle roof, which is typical for German houses. The living area is 150 m<sup>2</sup>. A core of six room types was developed to model the different rooms in the house: room types with two outer walls and room types with one outer wall. Some inner walls can face just one room, while others can face two rooms. Rooms on the ground floor are connected to the ground, while rooms on the upper floor have a saddle roof. The layout of the two floors is the same. The dimensions of the room are not fixed on a room level and are set up at floor level.

The package *House* contains the set up house models, where the room models are connected together. The name *OFD\_MiddleInnerLoadWall* denotes the fact that the standard house has a middle load bearing wall. Other positions of the load bearing inner wall are possible, but not included in the library. Walls are connected together and they form a room. Multiple rooms are connected together and they form a storey for the one family dwelling and an apartment for the multi-family dwelling. For the multi-family dwelling it is possible to model several storeys with apartments on top of each other, as well as several wings with apartments next to each other. Storeys are connected together to form a whole house. All connections between the connectors of the models are explicit.

For each standard house type there are packages for the building envelope (*Building*), for the energy system (*EnergySystem*) and for the building as a whole (*BuildingAndEnergySystem*). Currently work is be-

ing done on the models for the energy supply systems consisting of pump, heat generator, pipes, thermostatic valves and radiators. These models are being developed starting from more detailed models to be used as a teaching tool for a course on simulation of building energy systems. They will be made available later on as part of our contribution to the IEA Annex 60.

The package `Examples` contains exemplary simulation setups for a room, an apartment and a one family dwelling. They can be used to learn how to set up a simulation for these models (e.g. assumption for boundary conditions for a single room) and to compare the different CPU-times for simulations using the models.

## 2.2 Room model

In a room model the following physical processes are considered:

- transient heat conduction through walls; each wall consists of several layers with different physical properties; further discretization of each layer is possible
- steady-state heat conduction through glazing systems; transmission of short wave radiation through the window depends on a constant coefficient; transmitted radiation is considered together with the radiation from room facing elements
- heat convection at outside facing surfaces either with a constant coefficient, depending on wind speed, or depending on wind speed and surface abrasiveness
- heat convection at inside facing surfaces depends on the wall orientation and the temperature difference between the room air and the wall surface
- radiation exchange between room facing elements
- temperature balance equations for the room air volume; per room only one air node is considered; humidity is not considered in the air node

The incident solar radiation on tilted surfaces is calculated using a isotropic sky model [10].

All outer walls are whole walls connected to the room air and the ambient, while inner walls are half walls, each half belonging to one of the rooms which share the wall. Airflow among rooms is not explicitly considered.

## 2.3 Model parameterization

The room model is realized by aggregating together all the components in a model, parameterizing on a room level and referencing the parameter on the component level. In this way the number of parameters is reduced, e.g. for a simple rectangular room only three parameters are needed for the dimensions of all the walls: height, length, width. On a floor level this parameterization can further be optimized, as rooms have common walls.

However not the geometrical measurements are the most problematic when parameterizing a room, but the wall types, meaning their layer structure and the physical properties of each layer. We use records for parameterizing a wall. A record contains information about the number of layers, the thickness of the layer, as well as density, thermal capacity and conductivity of each layer material together with the emissivity of the room facing layer. However setting or changing the type of each wall in each room in a house can be challenging and can lead to errors. Because we aim to build standard house models, we want to parameterize a standard house with minimal but relevant input and not have to specify each wall individually. We chose to parameterize according to the following criteria:

- thermal mass class: heavy, middle and light
- energy saving ordinance: along with the already mentioned ordinance form 2009, older ordinances from 2002, 1995 and 1984 are considered

By specifying these two parameters, all wall, window and door types in a house are automatically set correctly.

In listing 1 an example is given of how to set up the parameter for the floor type depending on the set energy saving ordinance. As floor slabs are made of concrete for stability reasons, there is no difference between the floor types for houses with different thermal mass. As this type of coding is meant to help and not confuse a user, all these parameters are protected. The infiltration rate for a house is protected as well and depends on the energy saving ordinance. The *DataBase* library referenced in the listing is a library for records. The packages which are relevant for the *HouseModels* as well as any other relevant models, e.g. walls, windows, will also be made available.

Listing 1: Code example for setting the floor slab type (`Type_FL`) depending on the chosen thermal insulation

regulation (TIR)

```
// Floor to ground type
parameter DataBase.Walls.GFBaseDataDefinition
Type_FL=
if TIR == 1 then
DataBase.Walls.EnEV2009.Floor.FLground_SML()
else if TIR == 2 then
DataBase.Walls.EnEV2002.Floor.FLground_SML()
else if TIR == 3 then
DataBase.Walls.WSchV1995.Floor.FLground_SML()
else
DataBase.Walls.WSchV1984.Floor.FLground_SML()
```

Furthermore for a multi-family dwelling, for each apartment, the types for floor and ceiling are automatically set if the apartment is situated on the ground, last or an arbitrary upper floor.

## 2.4 Using and extending the library

We wanted to make the library easy to use and extend by future users and developers. To this purpose we put extra effort in creating easy to understand icons and graphical interfaces for parameter input.

Figure 2 shows the needed parameters for a room type with two outer walls. The room views integrated in the parameter window help the user understand which are the width and the length of the room, where each wall type is situated and what possibilities regarding windows and doors exist for the outer walls. For each outer wall both a door and a window are possible. By checking the box for a window, element input fields for parameters are enabled. In the given example the outer wall OW1 has a window and the outer wall OW2 has a door. The already mentioned parameters for thermal mass and energy saving ordinances are visible at the top level. Other necessary parameters include the solar absorptance coefficient of the walls and the heat convection model at outside facing surfaces.

However once the parameter window is closed, the information about the geometry of the room is no longer available. Because users might want to rotate or mirror a room to build up a whole floor, we wanted to transfer the information about the position of the walls in the room, the meaning of the parameters width and length as well as the existence of windows on the icon level.

Figure 3 shows the icon for the parameterisation from figure 2. The text for the connectors was added in post-processing. However the connectors themselves have unique, easy to understand names. The information about the width and the length of the room is fixed. The thickness of the pictures for the walls indicates if

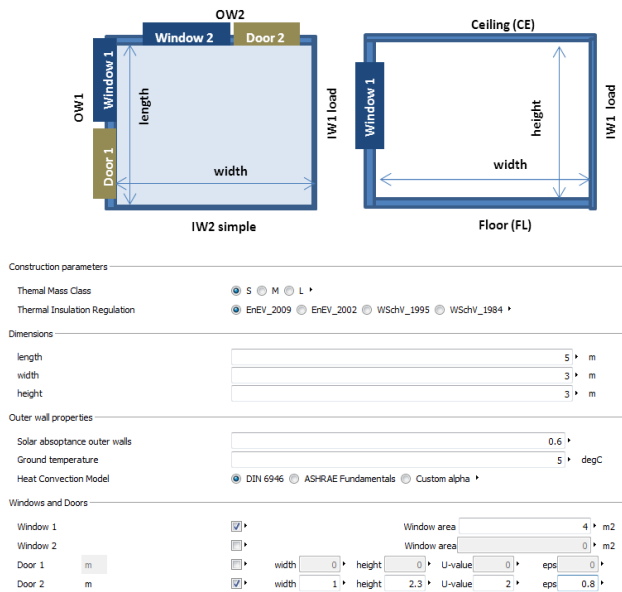


Figure 2: Parameter window for a room type

it is an inner or an outer wall. The square with the name *Win1* indicates the presence of a window on that wall, and the number *1* indicates that this is wall 1. The square is only visible if the window has been selected for the wall. In this way it is easy to combine the information given by the icon to the one which needs to be inputted in the parameter window.

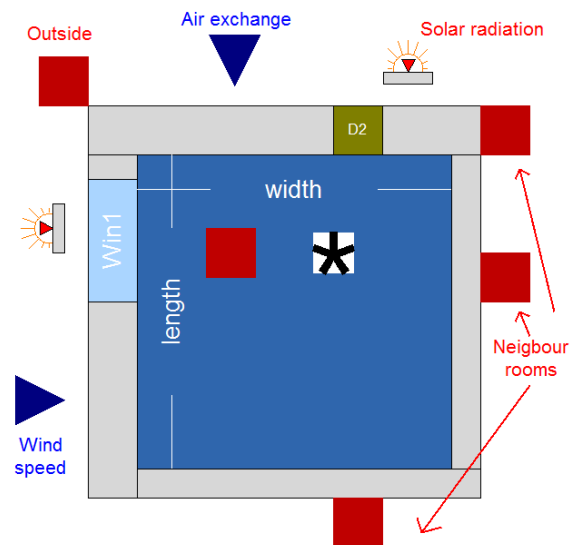


Figure 3: Icon for a room model

The set of room types developed for the one family dwelling can, if necessary, be parameterized differently than the standard model or extended in order to build up specific house models. New sets of wall, win-



dow and door types can be developed, e.g. for older, not renovated buildings, and incorporated in the existing structure.

With the exception of the integration of images in the parameter GUI, which is done by using a Dymola specific annotation, all other annotations are independent of the used simulation environment. Also the SI-units package from the Modelica Standard Library is sufficient for using the library.

We use this library in our teaching activities as well, as it is representative of our research activities and quite complex. With the help of a custom made tutorial we help students better understand working with Modelica from a building systems performance simulation point of view. The library is planned to be made available via a dedicated website in summer 2014.

### 3 Evaluation with Case 600 of ASHRAE Standard 140

The ASHRAE Standard 140 is widely used for the evaluation of building performance simulation software especially in the English speaking community. The test suite offers a variety of tests used for validation (base cases) and for evaluation and improvement of software tools (in-depth cases). In this paper we will present the results from using a room model build with our components in the simulation setup for case 600.

Case 600 is a case which tests a low mass building without shading. The building is a rectangular room with all surfaces facing to the outside, decoupled from the ground, with a height of 2.7 m, a length of 6 m and a width of 8 m. The south facing wall has a window area of 12 m<sup>2</sup>. Detailed information about the test is provided in [3]. Once the simulation results are calculated, the way to test them is by comparison with the simulation results from other software tools, which are provided in the standard. In the following subsections the results obtained with our room model are labeled *HouseModels Lib.*

In this paper we will presents results from evaluating the solar radiation model as well as the heating and cooling behavior of the room.

#### 3.1 Solar radiation model

The case offers a frame for evaluating the calculation of the solar radiation on a tilted surface, by providing results for solar radiation profiles for clear and cloudy days, for the south and west facing walls.

Figure 4 presents the solar radiation on the south side on a cloudy day, while figure 5 presents the results for a clear day. The profiles are similar to the ones obtained with the other programs, never going outside the maximal and minimal specified ranges.

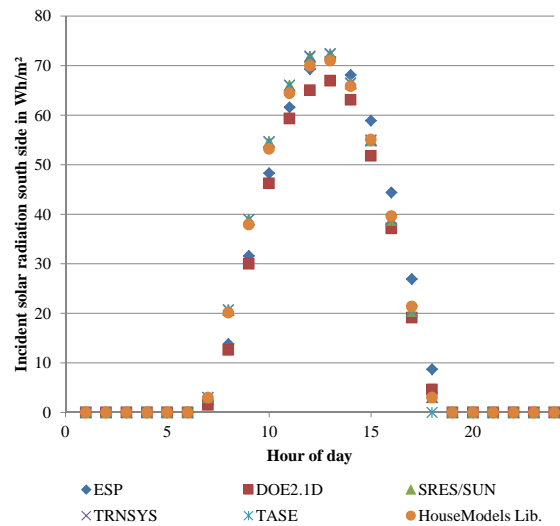


Figure 4: Comparison of hourly incident solar radiation for a cloudy day (March 5)

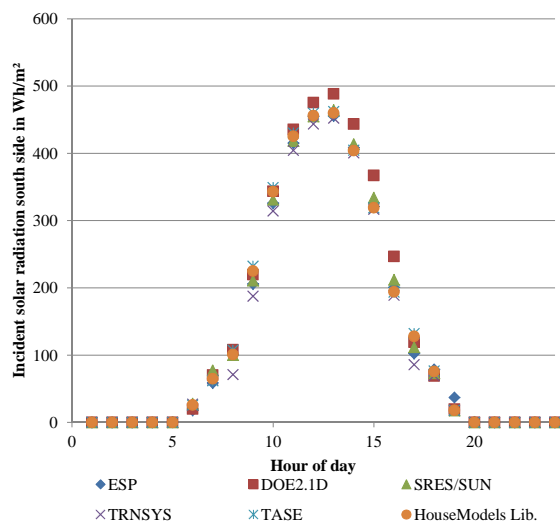


Figure 5: Comparison of hourly incident solar radiation for a cloudy day (July 27)

#### 3.2 Heating and Cooling behavior

For evaluating the heating and the cooling behavior of the room, annual heating and cooling loads and annual hourly peaks as well as profiles for exemplary days are calculated.

When considering the whole year, figure 6 presents a comparison of the heating and cooling loads for the whole year. The results for the annual heating load 5.081 MWh and cooling load 6.636 MWh are within the limits obtained with the other simulation tools.

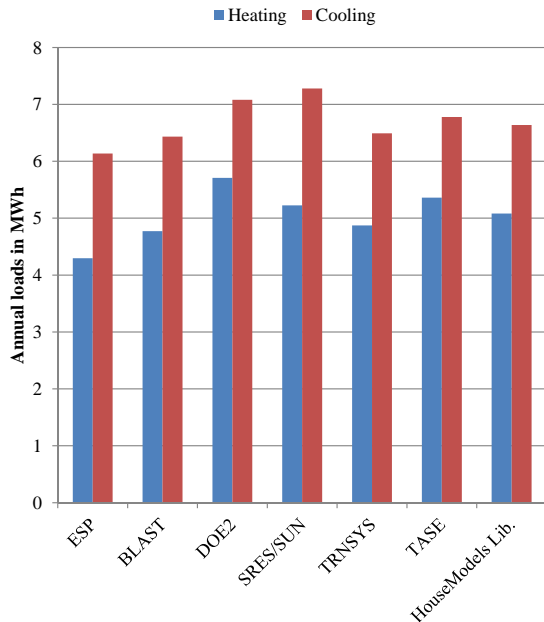


Figure 6: Comparison of annual heating and cooling loads

A more detailed look at the annual hourly heating and cooling peak loads (absolute value as well as day and hour when it occurs) is presented in tables 1 and 2. The peak heating load of 4.132 kW occurring on the 4<sup>th</sup> Jan and the peak cooling load of 6.2621 kW occurring on the 16<sup>th</sup> Oct are similar to the results from the other programs: they are within the given limits and there is at least one software which gives the same time of occurrence. The occurrence of the maximum cooling load on 16<sup>th</sup> Oct is explained by the concurrence of a high ambient temperature of 22.2°C and a high incident solar radiation on the south side of 874  $\frac{W}{m^2}$ .

Table 1: Annual hourly integrated peak heating loads

Code Name	kW	Date	Hour
ESP	3.437	4 JAN	5
BLAST	3.940	4 JAN	5
DOE2	4.045	4 JAN	5
SRES/SUN	4.258	4 JAN	2
TRNSYS	3.931	4 JAN	6
TASE	4.354	4 JAN	2
HouseModels Lib.	4.132	4 JAN	2

Table 2: Annual hourly integrated peak cooling loads

Code Name	kW	Date	Hour
ESP	6.149	17 OCT	13
BLAST	5.965	16 OCT	14
DOE2	6.656	16 OCT	13
SRES/SUN	6.827	16 OCT	14
TRNSYS	6.486	16 OCT	14
TASE	6.812	17 OCT	13
HouseModels Lib.	6.261	16 OCT	13

Finally the test requires a comparison of the heating and cooling load profiles for the day with the highest heating load. Positive loads are heating loads, while negative loads are cooling loads. As shown in figure 7 the results obtained with our room model are similar to the ones from the other simulation tools.

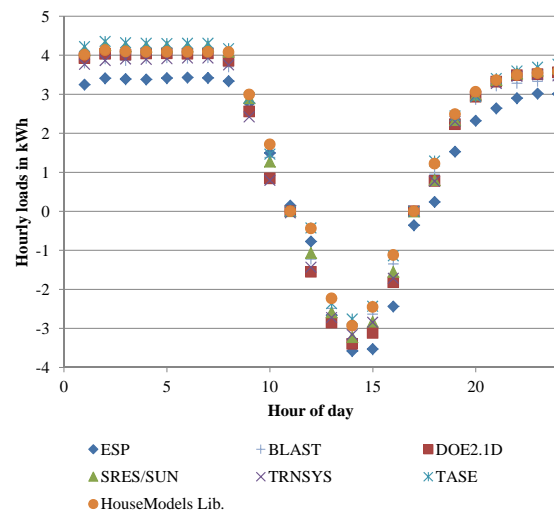


Figure 7: Comparison of hourly heating and cooling load profiles for Jan 4

## 4 Conclusion

In this paper we present our *HouseModels* library, which will be made available free of charge in summer 2014, as part of our contribution to the IEA Annex 60. The library contains complete standard house models for one and multi-family dwellings. The multi-family dwelling also contains a model for a single apartment. For each house type the models can be easily parameterized for different thermal masses and energy saving ordinances. A variation of the geometrical measure-

ments is possible for the one family dwelling. This type of variations are useful when testing energy concepts and control strategies, as a robust system has to be able to adapt to different types of buildings.

Our motivation for creating this library is to bridge the gap between developers and users of Modelica for dynamic building system simulations. The models are easy to understand and use. Extra effort has been made to enrich the parameter window and to make the icons dynamic in regard to the chosen parameters. In order not to confuse beginners certain parameters have been set as protected and the parameterization of a room model can be done by specifying only a handful of parameters. The library is a useful tool for teaching, as students can learn to use Modelica with a focus on building systems while at the same time learning a few programming tricks.

The set of room types developed for the one family dwelling can, if necessary, be parameterized differently than the standard model or extended in order to build up specific house models.

As we try to keep the models as simple as possible and as detailed as needed in order to have good CPU-times for the simulations, a validation of the room models is currently on the way. In this paper we present first results obtained with case 600 of the ASHARE Standard 140. For all the required outputs our room model produced results within the minimum and maximum specified ranges. We plan on further evaluating the models with the whole suite of tests and improving the models if necessary.

## References

- [1] Müller, D., Hosseini Badakhshani, A. Gekoppelte Gebäude- und Anlagesimulation mit Modelica, BauSim, Vienna, 2010.
- [2] IEA EBC Annex 60: New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards. Available: <http://www.iea-annex60.org/>.
- [3] ANSI/ASHRAE Standard 140: Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., Atlanta, 2007.
- [4] Wetter, M. Zuo, W., Nouidui, T.S. Recent developments of the Modelica buildings library for building energy and control systems, in *Proceedings of the 8th International Modelica Conference, Dresden, Germany, March 2011*, 2011.
- [5] Nytsch-Geusen, C., Huber, J., Ljubijankic, M. Modelica Building Systems - eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme, BauSim, Berlin 2012.
- [6] Nouidui, T.S., Phalak, K., Zuo, W., Wetter, M. Validation and application of the room model of the Modelica buildings library, in *Proceedings of the 9th International Modelica Conference, Munich, Germany, September 2012* 2012.
- [7] Dymola 2014, Dassault Systems, 2014.
- [8] Energieeinsparverordnung fuer Gebaeude (en: Energy Saving Ordinance), 2009.
- [9] Cali, D., Streblow, R., Müller, D., Osterhage, T. Holistic Renovation and Monitoring of Residential Buildings in *Proceedings of Rethink, renew, restart: ECEE 2013 summer study*, 2013.
- [10] Duffie, J., Beckmann, A. Solar engineering of thermal processes, Wiley, 2006.



# Modelica Library for Building and Low-Voltage Electrical AC and DC Grid Modeling

Juan Van Roy<sup>1,2,3</sup> Robbe Salenbien<sup>2,3</sup> Johan Driesen<sup>1,3</sup>

<sup>1</sup> University of Leuven, Department of Electrical Engineering (ESAT)  
Kasteelpark Arenberg 10, PB 2445, 3001 Leuven, Belgium

<sup>2</sup> Flemish Institution for Technological Research (VITO)

Unit Energy Technology, 2600 Mol, Belgium

<sup>3</sup> EnergyVille, 3600 Genk, Belgium

## Abstract

This paper presents a Modelica library for electrical grid systems for low-voltage distribution grids and in-building grids. The library is based on previous work, in which a library was presented to simulate fully balanced three-phase low-voltage distribution grids [1]. This library is extended to simulate three-phase unbalanced low-voltage distribution grids as part of the IDEAS library [2]. The library also allows to simulate in-building AC and DC grids. The AC grids can be single-phase or three-phase (un)balanced grids. Electrical grids may connect many different energy systems (loads and generation units), different grids and/or buildings within districts. The library allows to assess the grid impact of these systems on different grid types. Control or optimization strategies can use grid variables, such as voltages and power exchanges.

*Keywords:* Electrical grid; AC grid; DC grid; Power flow analysis; Modelica

## 1 Introduction

Climate and energy goals are set, i.e. the European 20/20/20 targets [3]. One of these targets is an improvement in the EU's energy efficiency and the integration of renewable energy resources (RES) in the power production. Also, energy goals and benchmarks at the level of individual buildings are stated in the European Directive 2010/31/EU [4]. It is stated that by 2020 all new buildings need to be nearly zero-energy buildings (nZEB). nZEBs target a high penetration of RES, such as photovoltaic (PV) systems, and a high energy efficiency in the built environment.

This integration of RES and energy-efficient technologies in buildings may result in an increased elec-

trification. Also different domains, such as the electrical and thermal domain, tend to become more integrated. This requires new approaches to analyze these integrated systems [5], such as taking into account the limitations of an electrical grid or using grid variables as control inputs in building design simulations. This may lead to a more effective analysis and better control of the energy system under consideration.

### 1.1 Electricity grids

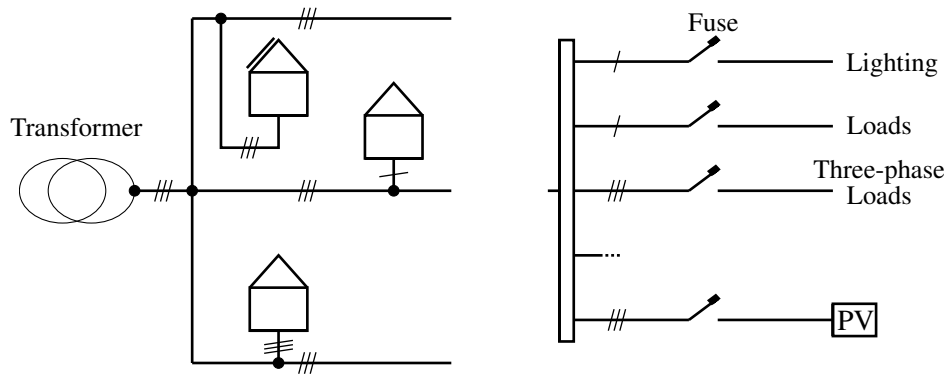
#### 1.1.1 Radial versus meshed grids

In the electrical power system, two types of electricity grids exist, namely distribution and transmission grids [6]. Distribution grids (low to medium voltage level) often differ fundamentally from transmission grids (high voltage level). Transmission grids are mostly meshed grids, whereas distribution grids are mostly radially. This means there is only one point of common coupling (PCC), which reduces the reliability of the distribution network. In case of a fault, all loads behind the fault will be switched off. Second, the R/X (resistance/reactance) ratio increases when the voltage level decreases. Thus, low voltage residential distribution grids are highly resistive.

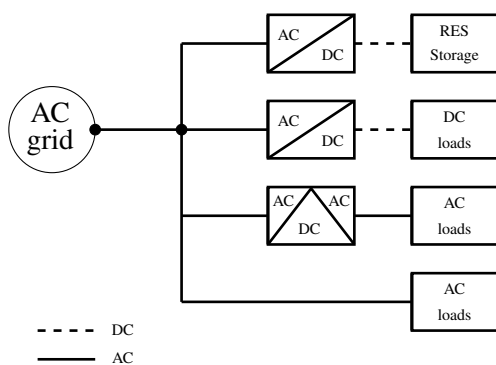
Electricity grids in buildings are similar to low-voltage distribution grids. As buildings are connected to the grid at one point, these grids are also radial. Loads (or buildings) can be single-phase or three-phase connected to the grid. Electrical grids in buildings can be a combination of different single and three-phase cables which connect the loads (see Fig. 1).

#### 1.1.2 AC versus DC in-building grids

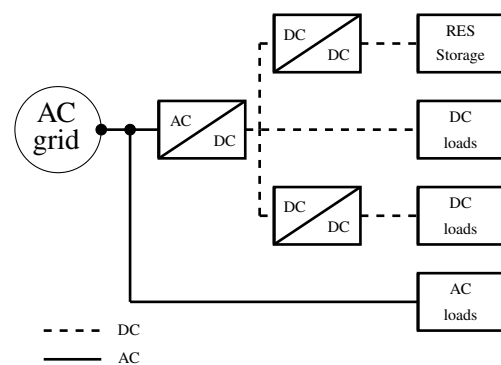
The current AC distribution grid and AC in-building grids are traditionally designed to feed AC and DC



**Figure 1:** Comparison of low-voltage distribution grids (left) and in-building grids (right).



**Figure 2:** A traditional AC in-building grid with distributed converters (e.g. in appliances).



**Figure 3:** A possible hybrid AC–DC grid with a central bidirectional converter (AC/DC).

loads (e.g. household appliances) with distributed converters, e.g. rectifier in an appliance (see Fig. 2). Since almost all electric loads are equipped with power electronic converters, the interest in DC grids increases. It may lead to efficiency and economic advantages by reducing the losses [7]. Also, the harmonic distortion can be significantly reduced, which saves energy and improves the lifetime of equipment [8]. And a central AC/DC connection with the distribution grid can help to solve the three-phase balancing problem. Fig. 3 shows a possible lay-out of a hybrid AC/DC electrical grid for buildings. This grid includes e.g. a central AC/DC converter, a central DC/DC converter for the DER, storage units, EVs, etc. and DC/DC converters for loads on different DC voltages.

## 1.2 Electricity grid impact

Energy efficiency can be achieved by a further electrification through new, more efficient technologies such as heat pumps and electric vehicles (EVs) [9]. However, from the electricity grid point of view, these new

technologies have a twofold grid impact.

First, the power consumption will increase. For instance, when full EVs are charged only at home, the additional power consumption of about 2,350 to 3,750 kWh doubles the average Flemish household’s power consumption. On the other hand, local RES will introduce bidirectional power flows due to a certain non-simultaneity with the local demand [10].

Second, the increased power consumption and the intermittent production character of RES and its potential non-simultaneity with the local power demand have an impact on the low-voltage electricity grid [10]: both the distribution and in-building grid. The injection of electricity and the increased power consumption may lead to peak loads and higher resistive losses. As LV grids are mainly resistive, voltage deviations and phase unbalance occur due to the active power flows [2, 10, 11]. To minimize the grid impact, a proper synchronization of consumption and production of electricity and heat is needed through demand side management (DSM), electrical and thermal sto-

rage and minimizing the power consumption [9].

Taking into account the limitations of an electrical grid or using grid variables as control inputs in building design simulations, may lead to a more effective analysis and better control of the energy system under consideration.

### 1.3 Scope of paper

To investigate the impact of energy systems on the electrical distribution grid or in-building grid, a library is developed to simulate both single and three-phase unbalanced AC radial grids and DC electrical grids. For AC grid analyses, a quasi-stationary model is implemented, assuming the frequency is fixed (e.g. 50 Hz). This allows to represent the waveforms by its amplitude and phase shift. Therefore, dynamic transient are not included.

In previous work, the first version of this library was presented [1]. This library consisted of models for the modeling of balanced three-phase low-voltage distribution grids, which can be represented by an equivalent single-phase grid. This paper discusses the recent additions: the unbalanced three-phase low-voltage grids and DC grids.

First, the basics on power flow analyses is given in Section 2. Section 3 and 4 describe the physical and the model description in Modelica. To conclude, a comparative model validation is performed.

## 2 Power flow analysis

A power flow analysis is performed to obtain the voltage and current information in each node and line of the electrical grid, based on the Laws of Kirchhoff:

**Conservation of electric charge** The sum of currents flowing into a node is equal to the sum of currents flowing out of a node.

**Conservation of energy** The sum of the voltage drops in any closed circuit is zero.

The voltage drop  $\Delta v$  in a line  $k$  between nodes  $n$  and  $n + 1$  is defined as:

$$\Delta v_k(t) = v^n(t) - v^{n+1}(t) = Z_k i_k(t), \quad (1)$$

with  $Z_k$  the impedance of the line and  $i_k$  the line current. When the nodal currents, line currents and nodal voltages are known, the apparent power in one phase  $S_f$  can be calculated.  $S$  consists of active power  $P$  and reactive power  $Q$ :

$$S_f(t) = P_f(t) + jQ_f(t) = v_f i_f^*, \quad (2)$$

with a non-linear relation between  $S_f$ ,  $v_f$  the phase voltage and  $i_f^*$ , the complex conjugate of the total phase current  $i_f$ . The total apparent power is calculated as  $S(t) = \sum S_f$ . For DC grids:  $Q = 0$ .

The joule losses  $P_J$  in a grid are the sum of the losses in all phases and neutral (or negative) conductor. The joule losses in a line  $k$  are calculated as follows:

$$P_{J,k} = R_k |i_k|^2, \quad (3)$$

with  $R_k$  the resistance of a line  $k$ . Note that also the reactive current is responsible for a part of  $P_J$ .

The non-linear system requires numerical methods to obtain a solution. Several methods are available to solve a power flow analysis, such as direct and iterative methods. The *backward-forward sweep* is an example of an iterative method, which is well suited for radial grids. This method is illustrated in Appendix A. In Dymola, the DASSL solver [12] is used to solve the power flow analysis.

In [13], a three-phase unbalanced power flow analysis is implemented in MATLAB. This model uses the backward-forward sweep technique. The models in the developed Modelica library will be validated with this MATLAB model. Other available Modelica libraries regarding electrical systems are e.g. the SPOT library [14] and Electric Power Library [15] which allow both steady-state and transient simulations.

The library in this paper allows to do a quasi-stationary analysis of electrical grids. This allows to represent the waveforms by its amplitude and phase shift. Dynamic transient are not included.

## 3 Physical model description

### 3.1 Grid topology representation

Traditionally, radial grids are represented by an incidence matrix (or connection matrix)  $\mathbf{T}$ . Eq. (4) gives an example of an incidence matrix of a grid in which consecutive nodes are connected.

$$\mathbf{T} = \begin{bmatrix} -1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix}. \quad (4)$$

The columns correspond with the number of nodes (or connection points), whereas each row is a segment of the grid (line) between two nodes. The start and end node of each line are represented by respectively

1 and -1. Since a radial grid contains  $n$  nodes, there are  $(n - 1)$  line segments. To attain a square matrix, an additional (first) row is introduced to represent the imaginary line segment between the transformer and the first node (with a length of 0 m). This line segment only has an end node.

## 3.2 Grid elements

### 3.2.1 Cables

Cables, with a length  $f_L$  (m), for the line segments are characterized by an impedance  $Z = R + jX$ , with  $R$  the resistance and  $X$  the reactance of the cable:

$$R = f_L r \quad \text{and} \quad X = f_L x, \quad (5)$$

with  $r$  the characteristic resistance and  $x$  the characteristic reactance in  $\Omega/\text{m}$ . This allows to represent the grid with an impedance matrix  $\mathbf{Z} = \mathbf{R} + j\mathbf{X}$ . Single-phase AC and DC grids contain two conductors, respectively a phase/neutral conductor and a positive/negative (or positive/neutral) conductor. Three-phase grids have three phases and one neutral conductor (total of four conductors).

For three-phase AC cables, typically nominal  $\Pi$  or  $T$  models are used to model cable, which include the resistance and reactance of the cable, as well as the shunt admittance. This capacitance is neglected for the considered low-voltage grids in this paper. Also, it is assumed that all phases are symmetrically spaced and that all phases are regularly transposed. Compared to [13], the shunt admittance and mutual impedance, which has a very small impact for low-voltage cables, is neglected.

### 3.2.2 Transformers

A transformer transforms e.g. a higher to a lower AC three-phase voltage level. The transformer is modeled with a phase impedance  $Z_{\text{tr}} = R_{\text{tr}} + jX_{\text{tr}}$ , which are assumed identical for the three phases. The losses in the transformer  $P_{\text{loss}}$  are the sum of the no-load losses  $P_0$ , which are assumed to be constant, and the sum of the joule losses  $P_j$  in each phase  $f$  of the transformer:

$$P_{\text{loss}} = P_0 + P_j = P_0 + \sum_{f=1}^3 R_{\text{tr}} |i_{f,k}|^2. \quad (6)$$

## 3.3 Load models

### 3.3.1 Loads and generation units

Loads and generation units are modeled as constant power loads, which is common for loads equipped

with power electronic. This means the power is not depending on the solution of the power flow (voltage and current). This results in a non-linear relation between power, voltage and current, as shown in Eq. (2) and its implementation in Section 4.2.4. Other possible load models take e.g. grid variables as input, such as voltage-droop load models [16].

### 3.3.2 Converters

Converters are required to convert AC to DC (rectifiers) or vice versa (inverters) or DC to DC. For rectifiers, the DC power ( $P_{\text{DC}}$ ) is lower (or equal) than the AC power ( $P_{\text{AC}}$ ). For inverters:  $P_{\text{AC}} \leq P_{\text{DC}}$ . The ratio between the DC and AC power is defined by the efficiency of the converter. The power electronics level of converters is not modeled.

Converters can also regulate the reactive power consumption or injection [13]. The power factor ( $\text{pf} = P/|S|$ ) can be leading (drawing reactive current) or lagging (injecting reactive current).

## 4 Implementation in Modelica

This section gives an overview on the functional requirements of the models, which are identified from the different use cases. In the second part, the component modeling is described.

### 4.1 Requirements

#### 4.1.1 Use cases

This section describes the needs for electrical grid modeling at the level of an individual building and at the level of a district energy system. The following use cases are identified, which are used to define the functional requirements:

- Grid impact analyses of electrical processes at building and district level. The results are an analysis of the nodal voltages, line currents, powers (active and reactive) and power losses.
- Grid architecture: single/three-phase AC grids and DC grids.
- Flexible and scalable approach for grid topology definition for all grid types.
- Connection architecture: Single/three-phase connection for loads and generation units.
- Integrated control or optimization by using grid variables, such as the nodal voltages, powers, etc.



### 4.1.2 Functional requirements

This section describes the functional requirements. Physics to be modeled include:

- Load and generation units:
  - Active and reactive power;
  - Connectors: Complex voltage/current and single/three-phase;
  - Converter losses.
- Electrical grids:
  - Quasi-stationary analysis;
  - Active and reactive power flows and losses;
  - Transformer: voltage drops and losses;
  - Unbalanced loads (including the resulting zero-point shifting);
  - Grid type:
    - \* Distribution and in-building grid;
    - \* Single-phase, three-phase (unbalanced) and equivalent single-phase;
    - \* AC and DC.
  - Integrated control or optimization: access to grid variables.

## 4.2 Modelica model implementation

### 4.2.1 Connectors

The library uses the connectors from the Standard Modelica Library:

- AC connectors: `Electrical.QuasiStationary.SinglePhase.Interfaces.Pin`
- DC connectors: `Electrical.Analog.Interfaces.Pin`

These connectors contain the voltage and current (flow). The AC positive and negative pin also include a reference angle.

The grid contains both *internal* and *external* nodes. The internal nodes include the neutral or negative connector, while the external nodes are used to connect the loads and generation units.

**External nodes** The external connection nodes are defined as: `.Pin Nodes[numPha, nNodes]`, with `nNodes` the number of grid nodes and `numPha=1` for single-phase AC and DC loads and generation units and `numPha=3` for three-phase AC loads and generation units.

**Internal nodes** The internal connection nodes are defined as `.Pin gridNodes[numCon, nNodes]`, with `numCon` the number of conductors (see Section 3.2).

**Adapters** Adapters are available to connect the internal and external nodes (see Code 1). For single-phase and DC grids, the adapter connects the two wire to a single wire system. For three-phase grids these are respectively four and three wire systems. A single-phase load or generation unit can be connected to one phase of a three-phase grid.

Fig. 4 shows the use of an adapter in a three-phase grid to connect the internal (`node4Lines`) and external nodes (`nodes3Ph`). The electricity grid connects the `gridConnection` (e.g. voltage source or transformer) with the internal nodes.

#### Two wires to single wire

```
twoWire[1].v - twoWire[2].v =
  oneWire[1].v "Phase voltage";
oneWire[1].i = -twoWire[1].i;
twoWire[1].i = -twoWire[2].i;
if AC then
  .Connections.branch(oneWire[1].
    reference,twoWire[1].reference);
oneWire[1].reference.gamma =
  twoWire[1].reference.gamma;
end if;
```

#### Four wires to three wires

```
for f in 1:3 loop
  fourWire[f].v - fourWire[4].v =
    threeWire[f].v "Phase voltage";
threeWire[f].i = -fourWire[f].i;
  .Connections.branch(threeWire[f].
    reference,fourWire[f].reference);
threeWire[f].reference.gamma =
  fourWire[f].reference.gamma;
end for;
fourWire[1].i + fourWire[2].i +
  fourWire[3].i = -fourWire[4].i;
```

Code 1: Adapters for internal and external nodes.

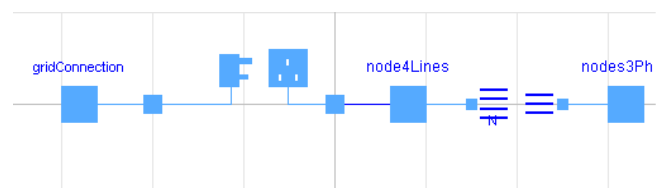


Figure 4: Use of adapter for three-phase grids.

### 4.2.2 Grid topology

The grid topology is described by the incidence matrix and the cable impedances (see Section 3). This is shown in Code 2. GridType extends GridImp and describes the grid topology.

Code 3 shows how the grid is constructed by using the incidence matrix. This code is used to construct each phase (and neutral) of the grid by connecting the different line segments (conductor) and grid nodes.

```

record GridType
  extends GridImp (R=CabTyp.RCha.*LenVec,
    X=CabTyp.XCha.*LenVec);
  parameter Modelica.SIunits.Length
    LenVec[nNodes] "Length for each line";
  parameter Cable CabTyp[nNodes];
end GridType;

record GridImp
  parameter Integer nNodes;
  parameter Integer nodeMatrix
    "Incidence matrix";
  parameter Modelica.SIunits.Resistance
    R[size(nodeMatrix,1)];
  parameter Modelica.SIunits.Reactance
    X[size(nodeMatrix,1)];
  parameter
    Modelica.SIunits.ComplexImpedance
    Z[size(nodeMatrix,1)] (re=R, im=X);
end GridImp;

record Cable
  parameter CharacteristicResistance RCha
    "Characteristic resistance";
  parameter CharacteristicReactance XCha
    "Characteristic reactance";
  parameter ComplexCharacteristicImpedance
    ZCha (re=RCha, im=XCha);
end Cable;
    
```

Code 2: Description of grid topology (interfaces).

### 4.2.3 Grid elements

**Cables** Code 2 also describes the record Cable. This record describes each cable type with their respective characteristic impedance, which is the impedance per unit of length ( $\Omega/m$ ).

**Transformers** The transformer model consists of an impedance for each of the three phases. The interface for the records, which are used to define the transformer data, is shown in Code 4.

```

For each conductor i
  phase + neutral/negative
  connect (internalNode[i], conductor[i].pin_p);
  for x in 1:nNodes loop
    for y in 1:nNodes loop
      if nodeMatrix[x,y] == 1 then
        connect (conductor[x].pin_p, node[2,y]);
      elseif nodeMatrix[x,y] == -1 then
        connect (conductor[x].pin_n, node[2,y]);
      end if;
    end for;
  end for;
    
```

Code 3: Connect statements for the grid construction.

```

record Transformer
  parameter Modelica.SIunits.ApparentPower
    Sn "Apparent power of the transformer";
  parameter Modelica.SIunits.ActivePower
    P0 "No-load losses";
  parameter Modelica.SIunits.Complex
    Impedance Z1 "Phase 1";
  parameter Modelica.SIunits.Complex
    Impedance Z2=Z1 "Phase 2";
  parameter Modelica.SIunits.Complex
    Impedance Z3=Z1 "Phase 3";
end Transformer;
    
```

Code 4: Transformer description (interface).

### 4.2.4 Load models

**Constant power model** Code 5 shows the non-linear relation between the apparent power, voltage and current shown in Eq. (2).

**Converters** The converters are implemented according to Section 3.3.2 and the power equations in Section 2. Fig. 5 shows the diagram for a bidirectional converter, which uses the AC and DC connectors as defined in Section 4.2.1. The AC connector can be single or three-phase. For three-phase converters, the apparent power is equally divided over the three phases:  $S_f(t) = S(t)/3$ .

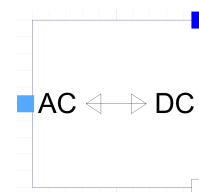


Figure 5: Use of adapter for three-phase grids.

```

model AC
  parameter Integer numPha;
  Modelica.Blocks.Interfaces.RealInput P;
  Modelica.Blocks.Interfaces.RealInput Q;
  Modelica.Electrical.QuasiStationary.
    SinglePhase.Interfaces.Negative
    Pin[numPha] vi
equation
  for f in 1:numPha loop
    P/numPha = Modelica.ComplexMath.real
      (vi[f].v*Modelica.ComplexMath.conj
      (vi[f].i));
    Q/numPha = Modelica.ComplexMath.imag
      (vi[f].v*Modelica.ComplexMath.conj
      (vi[f].i));
  end for;
end AC;

model DC
  Modelica.Blocks.Interfaces.RealInput P;
  Modelica.Electrical.Analog.Interfaces.
    PositivePin vi
equation
  P = vi.v * vi.i;
end DC;

```

**Code 5:** Relation apparent power, voltage and current for both AC (single/three-phase) and DC systems.

Code 6 shows the implementation of a bidirectional converter which extends the partial Converter model. A boolean is used to define the operation (rectifier or inverter) in function of the sign of the DC power. For the Rectifier and Inverter model, this boolean is a parameter.

```

model BidirectionalConverter
  extends .BaseClasses.Converters.
    Partials.Converter;
  Boolean inverter
  "Inverter: true / Rectifier: false";
equation
  inverter = if pDC >= 0 then true
    else false "Define converter mode";
  pAC = if inverter then -pDC*eff
    else -pDC/eff "DC/AC power ratio";
end BidirectionalConverter;

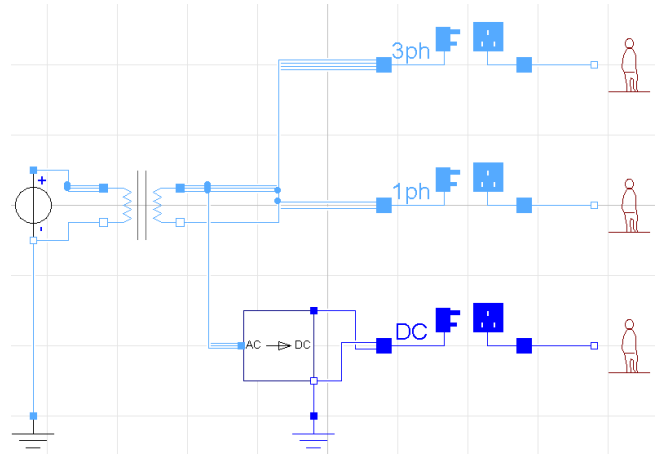
```

**Code 6:** Converter mode and DC/AC power ratio.

### 4.3 Simulation of multiple grid types

The Modelica implementation also allows to simulate multiple types of grid (i.e. single-phase AC, three-

phase AC and DC grids) in one simulation. Fig. 6 shows an example with different grids (with loads) connected to one feeding transformer. For the DC grid, a converter is required as well as a separate DC grounding.



**Figure 6:** Simulation of multiple grid types in one simulation.

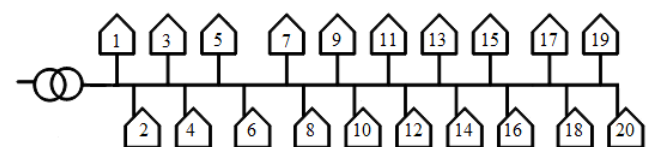
## 5 Comparative model validation

The electrical grid models are validated using a comparative validation method. The three-phase grid models from the Modelica library are compared with the power flow analysis tool in [13].

### 5.1 Example case: residential district

A simple case study is developed with a small three-phase unbalanced residential distribution grid.

There is one feeder connected to the feeding transformer. 20 residential loads are connected to respectively one of the 20 nodes in this feeder. The cables between the nodes have a length of 16 m and have a characteristic impedance of  $0.507 + j0.229$  m $\Omega$ /m. The feeder is connected to a three-phase transformer which has a phase impedance of  $20.4 + j67.5$  m $\Omega$ . The nominal phase voltage between a phase and the neutral conductor is 230 V. The grid topology is illustrated in Fig. 7 (not on scale).

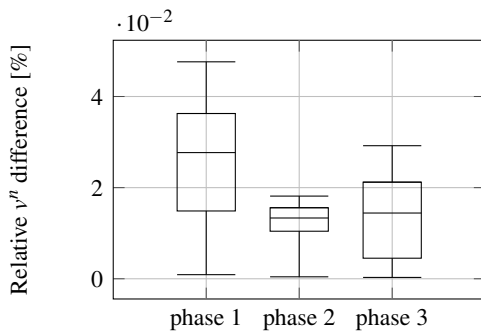


**Figure 7:** Grid topology for the validation model.

Synthetic residential load profiles for one day are available on a 30-minute resolution [17]. A set of 16 different profiles are randomly distributed over the different household connections to the distribution grid. The household loads are all single-phase connected.

## 5.2 Validation results

For the comparative validation, the results on the nodal phase voltages are compared with the results of the models from [13]. Fig. 8 shows the absolute<sup>1</sup> difference in results between the Modelica and Matlab models. Box plots are used to show the difference in the nodal phase voltages during the simulation period of one day for each phase of the electrical grid. The median, minimum and maximum values of these absolute differences are repeated in Table 1.



**Figure 8:** Relative absolute difference on the nodal phase voltages for each phase between the Modelica models and the Matlab models from [13].

**Table 1:** Median, average, minimum and maximum relative nodal voltage difference [%] for each phase.

	Median	Average	Min.	Max.
<b>Phase 1</b>	0.0277	0.0256	0.0009	0.0476
<b>Phase 2</b>	0.0133	0.0121	0.0004	0.0181
<b>Phase 3</b>	0.0144	0.0135	0.0003	0.0292

The average absolute nodal phase voltage differences for a voltage of 230 V are in the order of  $10^{-2}$  V. The minimum difference is less than 1 mV. Note that in this case study the nodal phase voltages are always lower than 230 V since there are only loads.

This difference between both models is function of the loads and the grid topology, since an error in one node will propagate through the grid as a result of the Laws of Kirchhoff in an electrical circuit. Therefore,

<sup>1</sup>The differences are both positive and negative

the difference in results will also increase for nodes further from the feeding point. A second source of the difference in results is the stop criterion for the iteration in both simulations. The Matlab code uses a maximum allowed voltage error (1 mV) as a stop criterion. In Dymola, a tolerance of  $10^{-4}$  is used. A last part of the difference is that the shunt admittance and mutual impedance between cables in three-phase systems is neglected in this Modelica library.

The differences between the results of both models are limited. Since both stop criteria for the iterative solution method is different, the results show that the accuracy is sufficient to apply this Modelica library.

## 6 Conclusions

A Modelica library is developed to simulate single-phase AC radial grids, both balanced and unbalanced three-phase AC radial grids and DC electrical grids. For AC grid analyses, a quasi-stationary model is implemented, assuming a fixed grid frequency (e.g. 50 Hz). The library is based on previous work, in which a library was presented to simulate fully balanced three-phase low-voltage distribution grids [1].

The models in this library can be used for electrical grid systems for low-voltage distribution grids and in-building grids. Electrical grids may connect many different energy systems (loads and generation units), different grids and/or buildings within districts. The library allows to assess the grid impact of these systems. Control or optimization strategies can use grid variables, such as voltages and power exchanges.

The comparative validation of this Modelica library with a power flow simulation in Matlab [13] shows that the difference in nodal voltages depends on the loads, the grid topology and end criterion. Also the mutual impedance of cables in three-phase systems is neglected. Nevertheless, the average voltage differences are limited, for this case study in the order of  $10^{-2}$  V for a voltage of 230 V.

## A Appendix: Backward-forward sweep method

The backward-forward sweep is an example of an iterative method to solve a power flow analysis. The method is well suited for radial grids.

First, an initial guess of the voltage profile is set. This allows the backward step to calculate the nodal ( $\mathbf{I}_{\text{node}}$ ) and line currents ( $\mathbf{I}_{\text{line}}$ ) in function of the ap-

parent power ( $\mathbf{S}_{\text{node}}$ ), the nodal voltages ( $\mathbf{U}_{\text{node}}$ )<sup>2</sup> and incidence matrix ( $\mathbf{T}$ ), which is defined in Section 3.1:

$$\mathbf{I}_{\text{node}} = f(\mathbf{S}_{\text{node}}, \mathbf{U}_{\text{node}}) = \left( \frac{\mathbf{S}_{\text{node}}}{\mathbf{U}_{\text{node}}} \right)^* \quad (7)$$

$$\mathbf{I}_{\text{line}} = (\text{transpose}(\mathbf{T}))^{-1} \cdot \mathbf{I}_{\text{node}}. \quad (8)$$

In the forward step, the nodal voltage is calculated with the line currents:  $\mathbf{U}_{\text{node}} = \mathbf{U}_{\text{grid}} - \mathbf{Z} \cdot \mathbf{I}_{\text{line}}$ . The iteration stops when convergence is reached.

## Acknowledgments

The work of J. Van Roy is funded through a doctoral scholarship of the Flemish Institute for Technological Research and the KIC InnoEnergy PhD School.

## References

- [1] B. Verbruggen, J. Van Roy, R. De Coninck, R. Baetens, L. Helsen, and J. Driesen, "Object-Oriented Electrical Grid and Photovoltaic system modelling in Modelica," in *8th International Modelica Conference*, Dresden, Germany, Mar. 2011.
- [2] R. Baetens, R. De Coninck, J. Van Roy, B. Verbruggen, J. Driesen, L. Helsen, and D. Saelens, "Assessing Electrical Bottlenecks at Feeder Level for Residential Net Zero-Energy Buildings by Integrated System Simulation," *Applied Energy*, vol. 96, pp. 74–83, Aug. 2012.
- [3] The European Commission. (2013, Dec.) The EU Climate and Energy Package. [Online]. Available: <http://ec.europa.eu/clima/policies/package/>
- [4] The European Parliament, "Directive 2010/31/EU of the European Parliament and the Council on 19 May 2010 on the Energy Performance of Buildings (recast)," 2010.
- [5] J. Van Roy, B. Verbruggen, and J. Driesen, "Ideas for tomorrow: New tools for integrated building and district modeling," *IEEE Power & Energy Magazine*, vol. 11, no. 5, pp. 75–81, Aug. 2013.
- [6] H. Lee Willis, *Power Distribution Planning Reference Book*. USA: Marcel Dekker, Inc., 1997.
- [7] P. Savege, R. Nordhaus, and S. Jamieson, "DC Microgrids: Benefits and Barriers." Yale School of Forestry & Environmental Studies.
- [8] B. Williamson, M. Redfern, R. Aggarwal, J. Allinson, C. Harris, P. Bowley, and R. Hotchkiss, "Project Edison: SMART-DC," in *IEEE PES Innovative Smart Grids Technologies Europe (ISGT)*, Manchester, United Kingdom, Dec. 2011.
- [9] International Energy Agency, "Energy Technology Perspectives 2012: Pathways to a Clean Energy System," 2012.
- [10] R. Passey, T. Spooner, I. MacGill, M. Watt, and K. Syngellakis, "The Potential Impacts of Grid-Connected Distributed Generation and How to Address Them: A Review of Technical and Non-Technical Factors," *Energy Policy*, vol. 39, no. 10, pp. 6280–6290, Oct. 2011.
- [11] K. Clement, E. Haesen, and J. Driesen, "The Impact of Vehicle-to-Grid on the Distribution Grid," *Electric Power System Research Journal*, vol. 81, no. 1, pp. 185–192, Jan. 2011.
- [12] L. Liu, F. Felgner, and G. Frey, "Comparison of 4 Numerical Solvers for Stiff and Hybrid Systems Simulation," in *IEEE Conference on Emerging Technologies and Factory Automation*, Bilbao, Spain, Sep. 2010.
- [13] J. Tant, F. Geth, D. Six, and J. Driesen, "Multiobjective Battery Storage to Improve PV Integration in Residential Distribution Grids," *IEEE Transaction on Sustainable Energy*, vol. 4, no. 1, pp. 182–191, Jan. 2013.
- [14] SPOT library. [Online]. Available: <https://github.com/modelica-3rdparty/SPOT>
- [15] Modelon. Electric Power Library. [Online]. Available: <http://www.modelon.com/products/modelica-libraries/electric-power-library>
- [16] F. Geth, N. Leemput, J. Van Roy, J. Büscher, R. Ponnette, and J. Driesen, "Voltage Droop Charging of Electric Vehicles in a Residential Distribution Feeder," in *IEEE PES Innovative Smart Grid Technologies (ISGT) Europe*, Berlin, Germany, Oct. 2012.
- [17] (2013, Dec.) Flemish Regulator for the Energy and Gas markets (VREG). [Online]. Available: <http://www.vreg.be>

<sup>2</sup> $\mathbf{I}_{\text{node}}$ ,  $\mathbf{I}_{\text{line}}$ ,  $\mathbf{S}_{\text{node}}$  and  $\mathbf{U}_{\text{node}}$  are all complex matrices.



# Tool coupling for the design and operation of building energy and control systems based on the Functional Mock-up Interface standard

Thierry Stephane Nouidui, Michael Wetter  
Lawrence Berkeley National Laboratory  
One Cyclotron Road, Berkeley, CA  
TSNouidui@lbl.gov

## Abstract

This paper describes software tools developed at the Lawrence Berkeley National Laboratory (LBNL) that can be coupled through the Functional Mock-up Interface standard in support of the design and operation of building energy and control systems. These tools have been developed to address the gaps and limitations encountered in legacy simulation tools. These tools were originally designed for the analysis of individual domains of buildings, and have been difficult to integrate with other tools for runtime data exchange. The coupling has been realized by use of the Functional Mock-up Interface for co-simulation, which standardizes an application programming interface for simulator interoperability that has been adopted in a variety of industrial domains.

As a variety of coupling scenarios are possible, this paper provides users with guidance on what coupling may be best suited for their application. Furthermore, the paper illustrates how tools can be integrated into a building management system to support the operation of buildings. These tools may be a design model that is used for real-time performance monitoring, a fault detection and diagnostics algorithm, or a control sequence, each of which may be exported as a Functional Mock-up Unit and made available in a building management system as an input/output block. We anticipate that this capability can contribute to bridging the observed performance gap between design and operational energy use of buildings.

*Keywords: Co-simulation; Functional Mock-up Interface; Building Management System; Niagara<sup>AX</sup>*

## 1 Introduction

Building thermal systems, ventilation systems, electrical systems and control systems are becoming more and more integrated to increase the energy effi-

ciency and to improve the interoperability with the electrical grid. This leads to a higher level of complexity for the design, installation, commissioning and operation of these systems. Modeling and simulation of such systems is challenging in today's simulation tools because it requires the tool to support multiple physical domains, multi-time scales, and also different formalisms for how systems evolve in time, in particular if they involve supervisory control with state transitions.

At present, the simulation of controls, rapid prototyping of new building energy and control systems and the use of simulation for building operations and building retrofits are constrained by current simulation tools. Most legacy whole building energy simulation tools such as EnergyPlus [1] or TRNSYS [2] perform well for annual energy analysis, but their model representation and numerical methods do not allow simulating systems with fast dynamics nor do they allow the proper representation of controls. For example, in EnergyPlus, the smallest time step is one minute, TRNSYS has a fixed time step, and neither can handle state events. To nevertheless use these tools with other programs that better address controls or systems with fast transients, but may lack comprehensive libraries of building components, they can be coupled with each other through co-simulation. By co-simulation, we mean a technique that allows individual component models described by differential algebraic or discrete equations to be simulated by different simulation tools running simultaneously and exchanging data during runtime.

Co-simulation somewhat remedies the limitations of individual tools. While this allows addressing many practical questions, see [3-5], one has to keep in mind that hybrid systems formed through this tool coupling still have some deficiencies. We refer to [6] for properties that would need to be satisfied by the individual tools to allow a proper treatment of hybrid systems.

This paper is structured as follows: Section 2 introduces the Functional Mock-up Interface (FMI) standard which is the open standardized interface used in this paper for co-simulation. Section 3 describes a) FMIs added to the Building Controls Virtual Test Bed (BCVTB), and EnergyPlus to support their co-simulation with various tools, and b) an FMI added to a building management system to support error-free development and deployment of control algorithms. Section 4 presents our conclusions.

## 2 Functional Mock-up Interface for Co-Simulation

The FMI standard has originally been developed in the Information Technology for European Advancement (ITEA2) project MODELISAR.

The FMI standard supports both model exchange and co-simulation of dynamic models using a combination of XML<sup>1</sup>-file, C-code and/or shared libraries.

The FMI standard version 1.0, which has been used in this contribution, consists of three parts:

- FMI for model exchange, which standardizes an interface for coupling simulation tools that are integrated in time by an external solver [7].
- FMI for co-simulation, which standardizes an interface for coupling simulation tools that contain their own solver for time integration [8].
- FMI for Product Lifecycle Management, which provides a standardized way to handle FMI related data [9].

A system model or simulation tool which implements the FMI standard is called a Functional Mock-up Unit (FMU). An FMU comes in the form of a zip-file, which contains the FMI model description file, which is an XML-file with information needed by an import tool, C-code and/or shared libraries required to interface with the model or simulation tool, and resource files such as tables, or documentation.

This contribution uses the FMI for co-simulation application programming interface (API). This API provides the means for two implementations namely *CoSimulation\_Tool*, and *CoSimulation\_StandAlone*. In the *CoSimulation\_Tool* implementation, the FMU contains a wrapper for shared libraries that interact with the slave tool so that a master tool which im-

ports the FMU can interface with the slave tool in a standardized way. In the *CoSimulation\_StandAlone* implementation, the FMU contains the model and its solver.

## 3 Co-simulation using the FMI Standard

### 3.1 FMU for Co-Simulation Import Interface in the BCVTB

The BCVTB [10] is a free, open-source middleware based on Ptolemy II [11]. It allows users to couple different simulation tools such as EnergyPlus, TRNSYS, MATLAB/Simulink [12], Modelica [13], or ESP-r [14] at runtime for co-simulation. The BCVTB also allows calling system commands, for example to run a shell script, which may start a Radiance-based [15] daylighting simulation. Figure 1 gives an overview of tools coupled to the BCVTB. The BCVTB also allows simulation tools to be coupled with hardware through its BACnet interface or its analog/digital interface [16].

The BCVTB is essentially a special configuration of Ptolemy II, with the addition of actors<sup>2</sup> and examples that are of interest to the buildings community.

The BCVTB has been used in several applications such as agent-based simulation [5], real-time simulation [3], controls of networked sensors and actuators [17], and performance prediction of HVAC systems [18].

As the BCVTB has been developed at the same time as the first version of the FMI standard, it contains its own API for co-simulation. This API is however much more limited than FMI and is not supported by all tools that export FMUs. Therefore, an FMU for co-simulation import interface has been added to the BCVTB. This interface allows the BCVTB to import simulation tools which have been exported as FMUs (Figure 2). This interface complies with the FMI for co-simulation API.

This new capability has several benefits:

- It allows users to couple the simulation tools shown in Figure 1, which are not all available as an FMU, to any other simulator that can be exported as an FMU for co-simulation.

---

<sup>1</sup>XML stands for Extensible Markup Language.

---

<sup>2</sup>Actors are software components that execute concurrently and share data with each other by sending messages via ports.



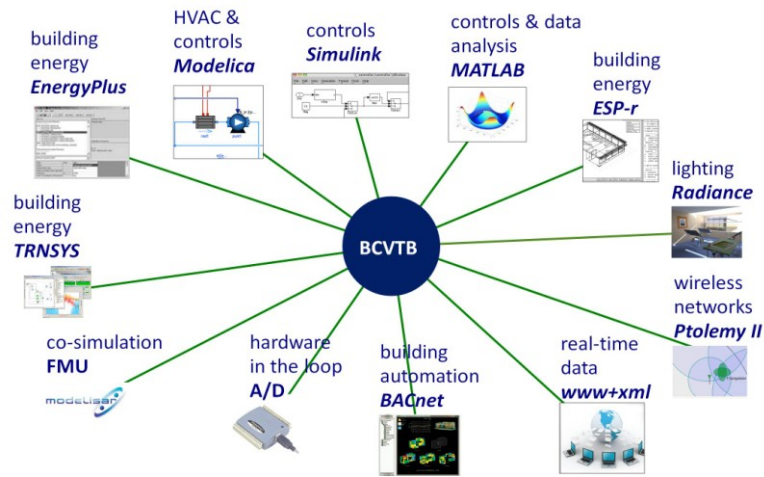


Figure 1: Simulation tools and hardware that can be coupled to the BCVTB.

- The BCVTB can be used as a master algorithm for co-simulation of FMUs using the Synchronous Data Flow domain of Ptolemy II.
- FMUs can be linked to hardware through the BCVTB.
- The BCVTB provides a graphical user interface for linking and simulating FMUs for co-simulation.
- The BCVTB allows synchronizing the simulation of FMUs to real-time.

could be a barrier for users who are familiar with one simulation tool and do not have resources to learn how to use this middleware. In addition, in some use cases, it may be more expedient to work directly in a domain-specific modeling environment. For example, when analyzing different façade systems, one may want to use a graphical user interface of a building simulation program that imports the model of the façade controller as an input/output block. Conversely, when developing a controller, one may want to take advantage of the visual editor and plotting capabilities of a Modelica modeling and simulation environment, while using an input/output block for a building model that takes as input the control action and outputs a sensor signal. For the first use case, we developed an FMU for co-simulation import interface in EnergyPlus, for the second use case, we developed a facility to export EnergyPlus as an FMU for co-simulation. The next two sections describe these technologies.

The use of the BCVTB has the drawback that it introduces an additional transaction layer between the different simulators. As computing time is for most applications dominated by the simulation code inside the FMUs, the BCVTB middleware generally has no noticeable effect on the computing time. However, users need to have some familiarity with the use of the BCVTB. This increases the learning curve and

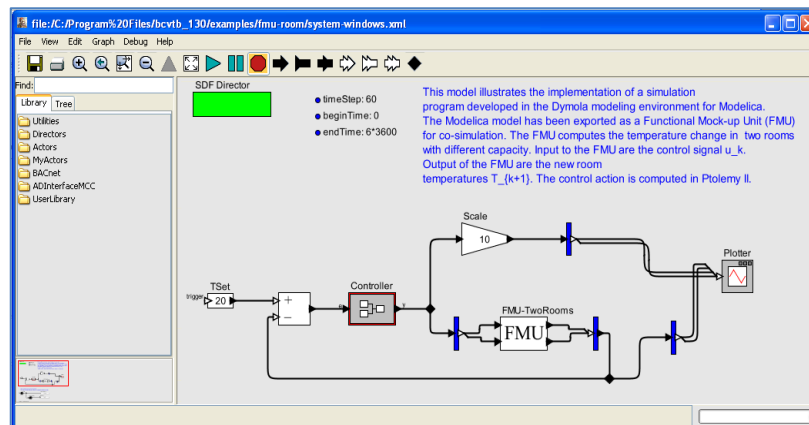


Figure 2: FMU for co-simulation import interface in the BCVTB.

### 3.2 FMU for Co-Simulation Import Interface in EnergyPlus

EnergyPlus is a whole building energy simulation tool. It is used by engineers, architects, and researchers for the modeling and simulation of energy use in buildings. EnergyPlus was not intended to be used for detailed modeling of airflow, dynamic response of heating, ventilation and air-conditioning equipment, and modeling of control systems other than scheduling of setpoints by simple supervisory control algorithms. To overcome these limitations, EnergyPlus has been coupled to various simulation tools such as COMIS [19], Computational Fluid Dynamics [20], MATLAB [21], or Modelica [22].

LBNL added to EnergyPlus 7.2 and higher an FMU for co-simulation import interface to allow the import of any simulation tool that is available as an FMU for co-simulation (Figure 3). This interface complies with the FMI for co-simulation API.

To facilitate the import of FMUs in EnergyPlus, we developed a utility called FMUParser. This utility is distributed with EnergyPlus and can be found in its *PreProcess* folder. When invoked, it unzips the FMU, extracts relevant information from the model description file of the FMU, and writes this information to a temporary EnergyPlus input file. A user can then complete this temporary input file to create the EnergyPlus input file. This parser has been developed so that users do not need to read the model

description file, which can easily contain more than thousand lines of xml syntax.

To support the import of FMUs, we extended the data structure of EnergyPlus with four new objects [23]. These objects are used to map the inputs and outputs of the FMU to internal EnergyPlus variables once the FMU has been imported in EnergyPlus. We also implemented a set of C-functions which are distributed with EnergyPlus as a shared library. EnergyPlus uses these functions to call the FMI functions of the imported FMU.

Figure 4 shows how the FMU for co-simulation import interface was used to couple an HVAC system, implemented in Modelica and exported as an FMU, to a room modeled in EnergyPlus. This example is described in detail in [24]. The HVAC system computed sensible and latent heat exchange with the room, using the air inlet and outlet as the thermodynamic boundary. The room model computed the temporal evolution of the room air temperature and humidity, using the sensible and latent heat exchange as inputs to its energy balance. The FMU uses as inputs the room dry-bulb temperature (*T<sub>RoomMea</sub>*), the outdoor dry-bulb temperature (*T<sub>DryBul</sub>*), the room air relative humidity (*rooRelHum*), and the outdoor air relative humidity (*outRelHum*) to compute the sensible and latent heat exchange (*Q<sub>Sensible</sub>*, *Q<sub>Latent</sub>*) which are sent to EnergyPlus through its outputs. EnergyPlus uses these values to compute the new room air temperature and humidity.

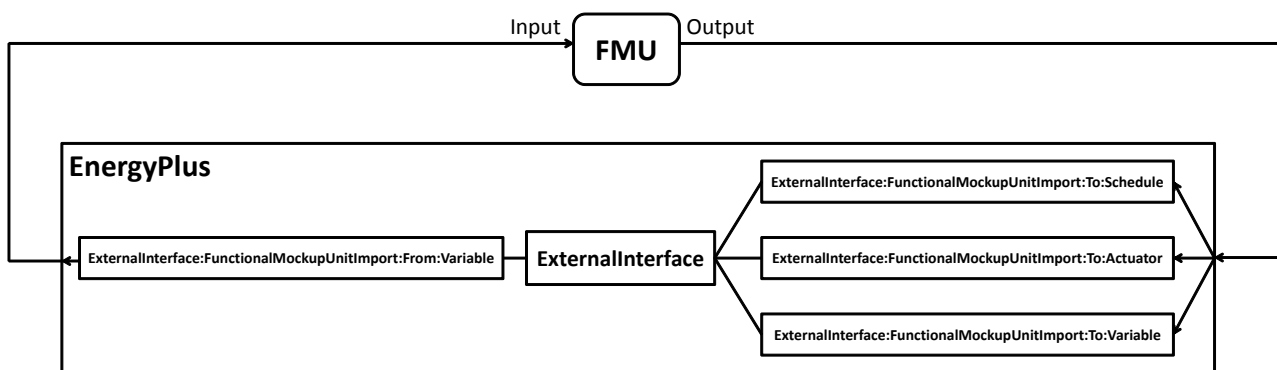


Figure 3: Importing an FMU for co-simulation in EnergyPlus through its ExternalInterface [25].

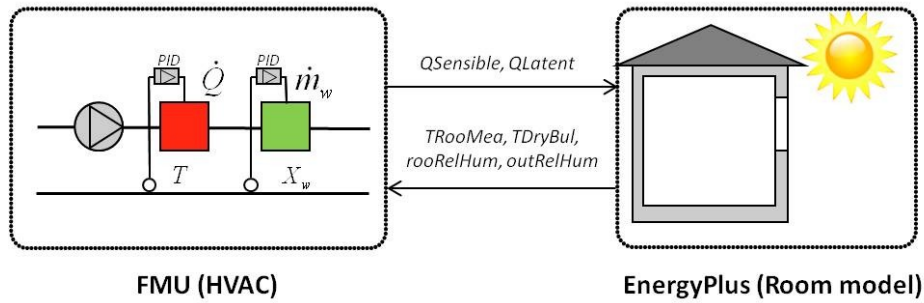


Figure 4: Linking an HVAC model developed in Modelica to an EnergyPlus room model using the FMU for co-simulation import interface.

### 3.3 FMU for Co-Simulation Export Interface of EnergyPlus

Although interfaces and middleware exist that facilitate the coupling of EnergyPlus with various software, they might not be widely used in the building simulation community since they still require users to be familiar with EnergyPlus so they can set it up and link it with other simulation tools. We thus exported EnergyPlus as an input/output block using the FMI standard. This allows importing EnergyPlus into any simulation tool that allows importing FMUs for co-simulation.

To export EnergyPlus 8.0 and higher as an FMU for co-simulation, we developed and released a software module called *EnergyPlusToFMU* [26], which exports EnergyPlus as an FMU for co-simulation. EnergyPlus implements in this configuration the FMI for co-simulation in the *CoSimulation\_Tool* method.

Figure 5 shows how EnergyPlus is imported in Dymola as an input/output block which can be connected to other Modelica blocks.

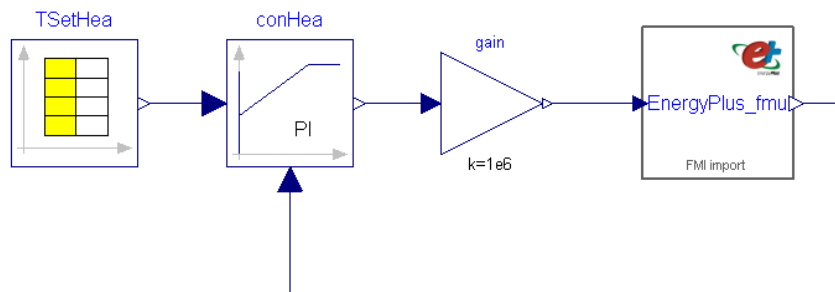


Figure 5: Coupling of an EnergyPlus model exported as an FMU with a PI-controller using Dymola.

To support the export of EnergyPlus as an FMU for co-simulation, we extended the data structure of EnergyPlus with four new objects [23]. These objects map the inputs and outputs of the FMU to variables that are internal to EnergyPlus.

Exporting EnergyPlus as an FMU for co-simulation can support various applications. For example, as described above, EnergyPlus may be used as an input/output block when designing a controller in a Modelica modeling and simulation environment or in MATLAB/Simulink. EnergyPlus building models may be linked to electrical grid and control models to design an electrical demand response controller for a campus that controls building electrical loads as a function of tariff, power quality and state of charge of batteries.

Table 1 shows a comparison between the different import and export capabilities. The table also lists their strengths and weaknesses. Although not exhaustive, this table should guide users in the selection of an import or export facility that is adequate for their specific application.

Table 1: Comparisons between different technologies to couple simulators<sup>3</sup>.

	Graphical user interface	Minimum knowledge required	Pros	Cons
BCVTB	Yes	BCVTB, and tools to be coupled.	Links to various tools and hardware.  Reuse functionality of Ptolemy II.	Learning curve, transaction layer.
BCVTB (FMU Import)	Yes	BCVTB and FMU.	Links to various tools, FMU for co-simulation and hardware.  Reuse functionality of Ptolemy II.	Learning curve, transaction layer.
EnergyPlus (FMU import)	No	EnergyPlus and FMU.	No need to learn new tool.  May be able to use graphical user interface of EnergyPlus <sup>4</sup> .	Tool to couple needs to be available as an FMU for co-simulation.
EnergyPlus (FMU export)	No	EnergyPlus and FMU.	EnergyPlus can be imported in other tools.  Can use EnergyPlus as an input/output block inside a block diagram editor.	Import tool needs to support FMI for co-simulation.

---

<sup>3</sup>Simulator refers to simulation tool, system model, or tool exported as an FMU for co-simulation.

<sup>4</sup>Not all graphical user interfaces of EnergyPlus may support all EnergyPlus features and thereby support the FMU import interface.

The previous sections described the coupling of multiple simulators. In the next section, we describe an import interface that we implemented in an open framework for building controls that allows linking FMUs for co-simulation with different building management systems.

### 3.4 FMU for Co-Simulation Import Interface in Niagara<sup>AX</sup>

Niagara<sup>AX</sup> is a Java-based framework and development environment for creating internet-enabled products, device-to-enterprise applications and distributed internet-enabled automation systems. It is a commercial product from Tridium that is often overlaid to other building management systems to facilitate their interoperability (Figure 6). Niagara<sup>AX</sup> uses a unified component model (Common Object Model) to transform the data from diverse external systems into uniform software components. These components form the foundation for building applications to manage and control the devices.

LBNL added to Niagara<sup>AX</sup> an FMU for co-simulation import interface. This interface complies with the FMI for co-simulation API.

We selected the Niagara<sup>AX</sup> framework because of its open-source architecture, which is based on Baja (Building Automation Java Architecture) [27] and its wide use in the buildings industry.

To implement the FMI interface in the Niagara<sup>AX</sup> framework, we used JFMI [28], a Java wrapper for FMI, and created two new classes `BFMUService` and `BFMUComponent`. These classes are used in the framework to interface with imported FMUs for co-simulation.

The `BFMUService` is used by the Niagara<sup>AX</sup> framework to process FMUs and to make their relevant information available to the framework. The `BFMUComponent` class represents an FMU instance. When instantiated, it appears in the

Niagara<sup>AX</sup> framework as an input/output block, which can be connected to other components of the Niagara<sup>AX</sup> framework.

Figure 7 shows an FMU for co-simulation which has been imported in Niagara<sup>AX</sup> as an input/output block. This block can then be linked to any other block available in the Niagara<sup>AX</sup> framework.

Adding an FMU for co-simulation import interface enables various applications. For example:

- An HVAC designer may create a simulation model during the design of a building. She/he then exports the model as an FMU for co-simulation and imports it to Niagara<sup>AX</sup>. In Niagara<sup>AX</sup>, she/he links the model input to measured data. The design model can then be used to compute expected energy consumption, which in turn can be used to compare measured with expected performance. See [3] for such a use case.
- A researcher or product developer may develop a fault detection and diagnostic algorithm, test it on a simulation model, and then export the algorithm as an FMU for co-simulation. This FMU can then be linked through Niagara<sup>AX</sup> with an actual building energy system.
- A researcher, product developer or advanced HVAC designer may develop and test an advanced control sequence in simulation, export it as an FMU for co-simulation, and import it to Niagara<sup>AX</sup> to link it to an actual building.

We anticipate that the use of FMI in building management systems supports a robust and low cost implementation, and an error-free deployment of controls or FDD algorithms.

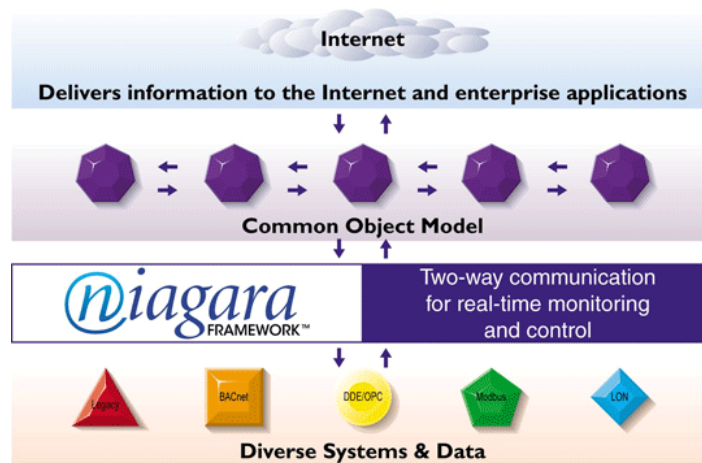


Figure 6: Niagara<sup>AX</sup> framework (Courtesy: Tridium).

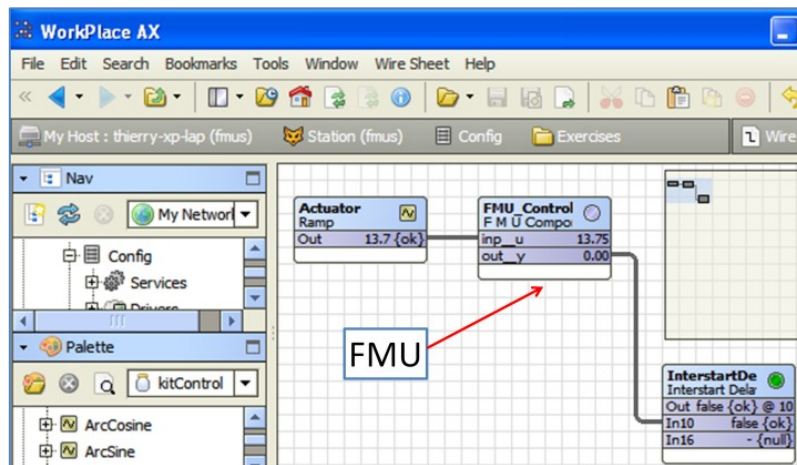


Figure 7: FMU for co-simulation import interface in Niagara<sup>AX</sup>.

## 4 Conclusions

We anticipate the integration of FMU for co-simulation interfaces in the BCVTB and EnergyPlus to support a better simulation-based design and operation of buildings.

We believe FMI to be well positioned to become a de-facto standard for implementing, and deploying control sequences. We thus see the integration of an FMU for co-simulation import interface in building management system as a promising approach and a natural extension of its application to date.

## Acknowledgments

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the US Department of Energy, under Contract No. DE-AC02-05CH11231.

We thank Pacific Controls for sponsoring the development of a Functional Mock-up Unit for co-simulation import interface in Niagara<sup>AX</sup>.

We thank Edward A. Lee, Christopher Brooks, David Broman, and Stavros Tripakis from the University of California at Berkeley for their support in developing JFMI and implementing the Functional Mock-up Unit for co-simulation import interface in the BCVTB.

## References

- [1] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, *et al.*, "EnergyPlus: creating a new-generation building energy simulation program," *Energy and Buildings*, vol. 33, pp. 319-331, Apr 2001.
- [2] A. Klein, J. A. Duffie, and W. A. Bechman, "TRNSYS - A transient simulation program," *ASHRAE Transactions*, vol. 82, pp. 623-683, 1976.
- [3] X. F. Pang, M. Wetter, P. Bhattacharya, and P. Haves, "A framework for simulation-based real-time whole building performance assessment," *Building and Environment*, vol. 54, pp. 100-108, Aug 2012.
- [4] D. Gyalistras, C. Sagerschnig, and M. Gwerder, "A Multi-stage Approach For Building And HVAC Model Validation And Its Application To A Swiss Office Building," in *13th International Conference of the International Building Performance Simulation Association*, Chambéry, France, 2013.
- [5] D.-W. Kim, J.-H. Kim, S.-L. Park, K.-C. Kim, and C.-S. Park, "Traditional Vs. Cognitive Agent Simulation," in *13th International Conference of the International Building Performance Simulation Association*, Chambéry, France, 2013.
- [6] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, *et al.*, "Determinate Composition of FMUs for Co-Simulation," in *Proc. of the International Conference on Embedded Software (EMSOFT 2013)*, Montreal, Canada, 2013.
- [7] MODELISAR-Consortium. (2010). *Functional Mock-up Interface for Model-Exchange*. Available: [https://svn.modelica.org/fmi/branches/public/specifications/FMI\\_for\\_ModelExchange\\_v1.0.pdf](https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_v1.0.pdf)
- [8] MODELISAR-Consortium. (2010). *Functional Mock-up Interface for Co-Simulation*. Available: [https://svn.modelica.org/fmi/branches/public/specifications/FMI\\_for\\_CoSimulation\\_v1.0.pdf](https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_CoSimulation_v1.0.pdf)
- [9] MODELISAR-Consortium. (2011). *Functional Mock-up Interface for Product Lifecycle Management*. Available: [https://svn.modelica.org/fmi/branches/public/specifications/FMI\\_for\\_PLM\\_v1.0.pdf](https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_PLM_v1.0.pdf)
- [10] M. Wetter, "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed," *Journal of Building Performance Simulation*, vol. 4, pp. 185-203, 2011.
- [11] E. A. Lee, S. Neuendorffer, and G. Zhou, *System Design, Modeling, and Simulation Using Ptolemy II*, Claudius Ptolemaeus ed.: Ptolemy.org, 2014.
- [12] Mathworks. (2013). *MATLAB & Simulink*. Available: [www.mathworks.com/](http://www.mathworks.com/)
- [13] S. E. Mattsson and H. Elmqvist, "An international effort to design the next generation modeling language," in *7th IFAC Symposium on Computer Aided Control Systems Design*, Gent, Belgium, 1997.
- [14] J. Clarke, "Moisture flow modelling within the ESP-r integrated building performance simulation system," *Journal of Building Performance Simulation*, vol. 6, pp. 385-399, Sep 1 2013.
- [15] A. Grynberg, "Validation of Radiance," Lawrence Berkeley Laboratory, Berkeley LBID 1575, 1989.
- [16] T. S. Nouidui, M. Wetter, Z. Li, X. Pang, P. Bhattacharya, and P. Haves, "BACnet and Analog/Digital Interfaces of the Building Controls Virtual Test Bed," in *12th International Building Performance Simulation Association Conference*, Sydney, Australia, 2011.
- [17] Y.-J. Wen, D. DiBartolomeo, and F. Rubinstein, "Co-simulation Based Building Controls Implementation with Networked Sensors and Actuators," in *BuildSys'11*, Seattle, WA, 2011.
- [18] M. Trcka, M. Wetter, and J. Hensen, "An implementation of co-simulation for performance prediction of innovative integrated HVAC systems in buildings," in *11th International Conference of the International Building Performance Simulation Association*, Glasgow, Scotland, 2009.
- [19] J. Huang, F. Winkelmann, F. Buhl, C. Pedersen, D. Fisher, R. Liesen, *et al.*, "Linking the COMIS multizone airflow model with the EnergyPlus building simulation program," in *Building Simulation 1999*, Kyoto, Japan, 1999, pp. 1065-1070.
- [20] Z. Zhai and Q. Chen, "Performance of coupled building energy and CFD

- simulations," *Energy and Buildings*, vol. 33, pp. 319-331, 2005.
- [21] W. Bernal, T. Nghiem, M. Behl, and R. Mangharam, "MLE+: A Tool for Integrated Design and Deployment of Energy Efficient Building Controls," in *4th ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Toronto, Canada, 2012.
- [22] C. Nytsch-Geusen, J. Huber, and Y. Nie, "Simulation-based design of PV cooling systems for residential buildings in hot and dry climates," in *13th International Conference of the International Building Performance Simulation Association*, Chambéry, France, 2013.
- [23] EnergyPlus. (2013). *Input Output Reference - The Encyclopedic Reference to EnergyPlus Input and Output*. Available: <http://apps1.eere.energy.gov/buildings/energyplus/pdfs/inputoutputreference.pdf>
- [24] T. S. Nouidui, M. Wetter, and W. Zuo, "Functional Mock-up Unit for Co-Simulation Import in EnergyPlus," *Journal of Building Performance Simulation*, vol. 7, pp. 192-202, 2013.
- [25] EnergyPlus, "External Interface(s) Application Guide, Guide for using EnergyPlus with External Interface," 2013.
- [26] T. S. Nouidui, D. M. Lorenzetti, and M. Wetter. (2013). *EnergyPlusToFMU*. Available: <http://simulationresearch.lbl.gov/fmu/EnergyPlus/export/index.html>
- [27] Tridium and Sun-Microsystems. (2000). *Baja: A Java - based Architecture Standard for the Building Automation Industry*. Available: [http://www.automatedbuildings.com/wsim/Baja\\_White\\_Paper.pdf](http://www.automatedbuildings.com/wsim/Baja_White_Paper.pdf)
- [28] C. Brooks, E. A. Lee, M. Wetter, T. S. Nouidui, D. Broman, and S. Tripakis. (2012). *JFMI - A Java Wrapper for the Functional Mock-up Interface*. Available: <http://ptolemy.eecs.berkeley.edu/java/jfmi/>



# Coupling occupant behaviour with a building energy model - A FMI application

Gilles Plessis<sup>1</sup>

Édouard Amouroux<sup>2</sup>

Yvon Haradji<sup>1</sup>

<sup>1</sup>EnerBaT – EDF R&D, Moret sur Loing, FRANCE

<sup>2</sup>LIP6 – Université de Paris 6, Paris, FRANCE

gilles.plessis@edf.fr

edamouroux@gmail.com

yvon.haradji@edf.fr

## Abstract

This paper illustrates the use of *Functional Mock-up Interface* (FMI) [1] to couple an occupant behaviour simulator and a building model.

Due to their intrinsic nature, occupant behaviour and building and its energy systems are usually represented by different modelling paradigms. The occupant behaviour is here described by Agent-Based Modelling (ABM) whereas the building is described by a set of hybrid and differential algebraic equations, typical of dynamic thermal modelling. Such different complex systems cannot be efficiently simulated in a single tool. Therefore, one solution is the tool coupling approach.

The FMI standard for co-simulation was used to couple the SMACH occupant behaviour simulator and a building energy model built with the BuildSysPro Modelica library. Variables of interest are passed from one model to another at fixed synchronization time steps.

*Keywords: Building simulation; behavioural modeling; Specific use of electricity; thermal comfort; Modelica; FMI; co-simulation*

## 1 Introduction

Enforcement of energy efficiency policies drives new buildings towards better performance and especially low or even positive energy buildings. These kinds of buildings are different from existing ones as their ventilation and envelope heat losses are intensively decreased. Nevertheless, this improvement has two consequences. First, human actions to ensure thermal comfort can cut back heating and cooling energy savings. Second, a significant share of energy will be consumed by specific electricity uses (light-

ing, cooking, white goods, electronic appliances...) and not anymore by space heating and cooling. Both points are strongly linked with occupant behaviour.

Occupant behaviour is commonly described in dynamic building simulation tools using standardized occupancy profiles. Various studies suggest that occupant behaviour should be taken into account in a more accurate way, as it can have a dramatic impact on energy consumption especially in the context of low and positive energy buildings [2] [3].

The purpose of this work is to couple realistic occupant behaviours with building energy simulation. It focuses on dynamic modelling and especially electric power demand instead of energy consumption. Two interaction approaches were carried out:

- Co-simulation for R&D studies,
- Generation of realistic occupancy scenarios for simplified building simulation tools.

This last objective will be fulfilled thanks to design of experiments using the co-simulation. Only the tool-coupling approach is presented here.

## 2 Occupant behaviour model

The occupant behaviour model is implemented in the SMACH platform, an agent-based tool developed by EDF and the LIP6 laboratory.

The coupling between the building energy model and the occupant behaviour model is mainly carried out through thermal comfort perception and control of electrical appliances.

### 2.1 Occupant behaviour model

In order to represent adequately and individually the behaviour of each occupant, we rely on an agent-based modelling approach and, more specifically, the

one we already applied in [4]. The agents, in SMACH, provide a realistic way of modelling inhabitants' behaviour. In the occupant behaviour model, an agent corresponds to an individual person, and a family corresponds to a group of agents. This modelling approach emphasises the fact that agents are autonomous and have their own individual interaction dynamic within the situation. At each moment, their universe is defined by their individual perception of the situation, their comfort level and their preferences regarding possible actions [5]. State-graphs could have been used however this kind of approach is much less adaptable and concise than ABM for occupant behaviour modelling.

In practice, the inhabitants' activity is decomposed into generic tasks ( $t$ ) such as "watch TV" or "cook dinner". The actual behaviour of each individual is a set of actions ( $a$ ) derived from the generic tasks. A task is a tuple  $(\tau_{\min}, \tau_{\max}, E_t, E_{tf}, T_{pre})$  corresponding respectively to minimal and maximal duration to conduct the task, the required and favourable appliances and the pre-conditional tasks. For instance, ironing may require the completion of cleaning clothes, lasts one to two hours, requires the iron appliance and may be likely to be conducted with the TV switched on.

An action is an instantiation of a task by an occupant. It is defined by a tuple  $a = (t, w, st)$  corresponding respectively to the associated task, rhythm and its state (done, not-done). A rhythm ( $w$ ) is a tuple  $(per, freq, var, PP)$  corresponding respectively to the base period (day, week, month, year), the frequency, the frequency variability and a set of preferred periods (PP). An action example could be as follows: a child may watch TV according to a weekly rhythm of 10 viewing periods on weekdays between 7 and 8 a.m. and/or 5 and 6 p.m.

## 2.2 Thermal comfort model

The thermal comfort model used in the occupant behaviour simulator is derived from Fanger's PMV model [6]. Instead of defining a mean comfort value, an individual thermal comfort level is defined after the same set of variables (air temperature, radiative temperature, humidity, metabolic heat production and mechanical work). A new parameter called *frilosity* defined by expert assessment and based on field studies in real situation is also taking into account. It describes the cold tolerance of each occupant [7]. On top of this individual model, a group comfort level is defined per room as follows:

$$GC = \sum_{i \in n} \frac{comfort_i \cdot age_i}{n}$$

Where  $age_i$  is the age class and  $comfort_i$  the individuals comfort. The age class defines a level of responsibility depending on age, for instance an adult will favour a child comfort instead of his/her. This group comfort level is used to determine what action the group will choose (e.g. increase temperature set point, open windows) and individuals' actions (e.g. adapt clothing, change activity ...).

## 2.3 Appliances

Electrical appliances,  $e \in E$ , are defined by their electrical power consumption  $\theta_i$ . The *power* function defines this relation for each appliance.

$$power : \begin{cases} E \rightarrow \mathbb{R}^+ \\ e \mapsto \theta_i \end{cases}$$

We consider two types of electrical appliance: *state-based appliances* and *program-based appliances*.

*State-based appliances* are defined as a tuple  $(\theta_o, \theta_s, st)$  where  $\theta_o$  and  $\theta_s$  are the electrical power consumptions when  $e$  is running or in standby mode, and  $st \in \{\text{off, standby, on}\}$  is the state of the appliance, modified by occupants in the house during their activities. Heaters, TVs, fridges... can be represented by state-based appliances.

*Program-based appliances* are defined as an ordered pair  $(P_e, st)$  where  $P_e$  is a set of operating programs characterised by load curves. The status  $st$  of the appliance is then defined by an ordered pair  $st = (p, t)$  where  $p$  is the currently selected program and  $t$  is the time since the beginning of this program. A program  $p$  is an ordered pair  $(\tau, \varphi)$  where  $\tau$  is the program duration and  $\varphi: [0, \tau] \rightarrow \mathbb{R}^+$  gives the appliances power consumption over time during the program. Thus,  $power(e) = \varphi_p(t)$ . For instance, washing machines can be represented by this kind of appliance.

In SMACH, all energy consumption profiles come from real data from the REMODECE European project [8].

## 3 Building energy model

The building energy model is written in Modelica language with the BuildSysPro library developed by EDF [9]. We used a purely thermal model compliant with the *Thermal.HeatTransfer* class from the Modelica standard library. The class is defined by its connector involving the temperature  $T$  as a potential and  $Q\_flow$  for the heat flow rate.

### 3.1 Building envelope

The “Mozart” house is one of the most representative houses in the French residential building stock, and was therefore chosen for this work. It is a medium size detached house of 100 m<sup>2</sup> of living surface area and an air volume of 252.15 m<sup>3</sup>.

The building is modelled in a low energy configuration: the U-values of the different envelope components are low compared to the standard French building stock. Internal wall insulation is chosen since this is the most common in France; it impacts the potential of heat storage into walls. The internal walls are modelled in the same way as the other opaque walls. Therefore, they contribute to the thermal inertia.

**Table 1: Main parameters of the building envelope**

PARAMETER	VALUE	UNIT
U <sub>outdoor wall</sub>	0.27	[W.m <sup>-2</sup> .K <sup>-1</sup> ]
U <sub>ceiling</sub>	0.191	[W.m <sup>-2</sup> .K <sup>-1</sup> ]
U <sub>floor</sub>	0.263	[W.m <sup>-2</sup> .K <sup>-1</sup> ]
U <sub>windows</sub>	1.43	[W.m <sup>-2</sup> .K <sup>-1</sup> ]

Windows are double-glazed, with no thermal inertia. The conductive, convective and radiative heat transfers are considered.

The building envelope model is composed of six zones corresponding to the different rooms represented in Figure 1. The garage is not taken into account.

### 3.2 Boundary conditions

Weather data is applied as boundary conditions on the outdoor side of the building model. The building envelope is studied in a temperate climate, more precisely the weather data from Trappes, a city lo-

cated near Paris in France. The weather reader model provides the outdoor dry air temperature, the direct and diffuse solar radiations and the sky temperature. Relative humidity and wind data (velocity and direction) are not used in this first study.

The short wave radiations transmitted inside the building envelope through the windows are entirely absorbed by the floor. The long wave radiative heat transfers which occur between the sky and surroundings and the external surfaces of the walls and windows are taken into account through a combined heat transfer coefficient.

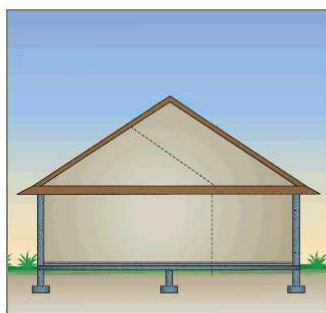
### 3.3 Model inputs

Internal heat gains due to the occupants are set to 90 W.pers<sup>-1</sup>. The indoor occupancy scenario is coupled to the occupant behaviour model, impacting internal gains and set point temperatures, which are specified in each zone by the occupants depending on their thermal comfort perception. Occupants also have the possibility to open windows if needed to ensure their comfort. The window open/closed status has an impact on the internal gains by adding a heat gain or loss, depending on the external weather conditions.

### 3.4 HVAC system

The HVAC system is composed of an ideal electrical heater controlled by PID and a mechanical ventilation. A static model is used for the ventilation system with a fixed air change rate.

Considering the weather conditions of Trappes and a normative scenario for occupancy specifying internal heat gains, the annual heat demand for this building is 20 kWh.m<sup>-2</sup>.year<sup>-1</sup>.



**Mozart**



**Figure 1: Diagrams of the Mozart house**

Figure 2 shows the overall Modelica model.

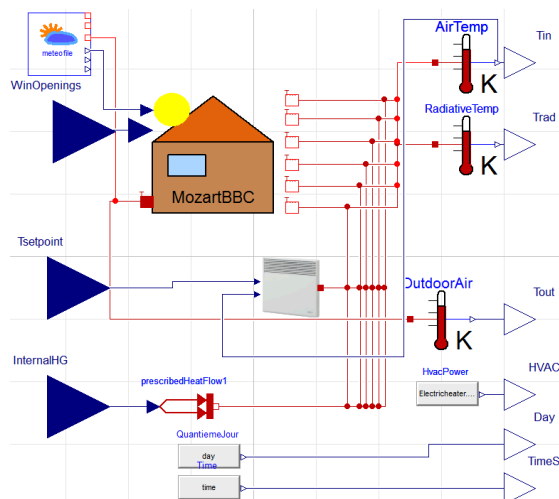


Figure 2: Overall building energy model

## 4 Co-simulation

As previously mentioned, the model describing the building and its energy system has been developed using the BuildSysPro library with Dymola 2014. The occupant model has been implemented within the SMACH agent-based simulation platform, written in Java. In order to allow co-simulation of heterogeneous simulators, the Modelica building model was exported from Dymola as a *Functional Mock-up Unit* (FMU), based on the *Functional Mock-up Interface* (FMI) for co-simulation standard. This approach has already been applied in several works [10] [11].

### 4.1 Implementation

FMI defines a standardized interface to be used in computer simulations to develop complex systems [1]. In practice, it defines a common interface to control the FMU as a dynamic library, a *.dll* file in the present case. The first version of the standard was used.

The master algorithm part is integrated into the SMACH occupant simulator. The JFMI wrapper [12] was used to control the exported FMU within the SMACH platform. It is designed to facilitate control of a native library with Java. A centralised architecture was used with a single FMU representing the entire building with its energy systems. The SUNDIALS CVODE solver with backward differentiation formula [13] was integrated as the built-in solver of the FMU.

### 4.2 Coupling variables

Several coupling variables are selected in order to couple the building energy model and the occupant simulator. On one hand, the occupant simulator must supply data that can impact the thermal ambiance of each room in the building or the HVAC system. On the other hand, the building energy model must return information to estimate the overall power load and the thermal comfort of the occupants. Therefore the FMU inputs are:

- The temperature set point of each room where occupants can set the temperature.
- The internal heat gains due to appliances and occupancy.
- The windows opening status.

The FMU outputs are:

- The air temperature in each room.
- The mean radiative temperature in each room.
- Outdoor temperature.
- Electrical power consumption of the HVAC system

There are also two optional outputs to ensure proper synchronisation:

- Day
- Time in second

### 4.3 Workflow

The master algorithm acts as a scheduler between the FMU and the occupant model within the SMACH platform. We use a constant synchronization time step of 1 minute. After the instantiation and initialisation steps, the workflow is the following:

1. The behaviour of each occupant is computed based on his/her perceptions (action of other individuals, temperature of the room...) and internal state (action that he yet has to perform, thermal comfort level...).
2. The FMU inputs corresponding to occupants' actions on thermal environment are set (heater control, opening of the windows...).
3. The building energy model computes the actual temperatures based on the occupants actions and their presence (internal heat gains)
4. The FMU outputs are returned to the occupant simulator.

### 4.4 Computing performances

On a computing time point of view, the coupling is heavy. For a one month simulation and a time step of 1 minute, the occupant behaviour simulation takes 3 minutes with SMACH and the building energy simulation takes 10 seconds with Dymola. When coupling

is applied, the same simulation takes 13 minutes. These computing times have been obtained with standard laptop equipped with an Intel i5 2520 M processor on Windows 7 32 bits. These are rough results and no optimisation has been conducted yet.

## 5 Results

This section shows some outputs from the co-simulation between SMACH and the BuildSysPro building model.

The activity diagram in SMACH, Figure 3, is used to analyse simulations. Activities are represented by different colours for each occupant over time.

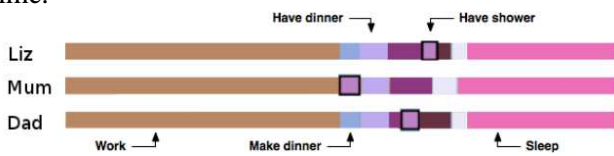


Figure 3: Activity diagram, a working day pattern

On Figure 4, the activity diagram clearly shows the difference between usual working days and other days (Wednesday and weekend) for this household. Figure 5 shows the temperature evolution over thirteen days including seven days of vacation. The blue line represents the outdoor temperature and the red line, the indoor temperature in the living room. The grey stripes show when the HVAC system is working.

In fact, the actual temperatures and power consumption curves differ because during regular periods, the

family is not at home during weekday daytime and sets the objective to 18°C whereas the indoor temperature is set according to each individual/group comfort level when they are at home. On the contrary, during holidays the indoor temperature is set at 12°C thus, the electric heater power load is null for an extended period of time (due to the efficiency of the represented low energy house) and then can oscillate to maintain this temperature. One may also observe the difference between weekdays and weekends. The absence of occupants during weekdays lets the HVAC system controls the temperature according to the temperature set points.

## 6 Conclusion

This paper presents the first results of the co-simulation between the SMACH platform for occupant behaviour and a BuildSysPro building model. The tool coupling is fully functional, however to ensure computer time efficiency and adaptability the following improvements will be considered in IEA Annex 60 [14]:

- Control of the communication step size,
- Dividing the Modelica model into sub-models (HVAC system, building envelope) and use composition of FMUs to ensure adaptability,
- Auto-mapping of coupling variables, considering input and output names and dimensions,

The authors would like to thank the ANR for funding the SUPERBAT project.

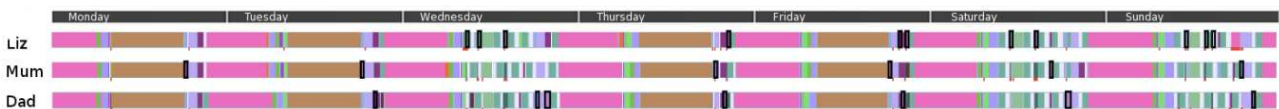
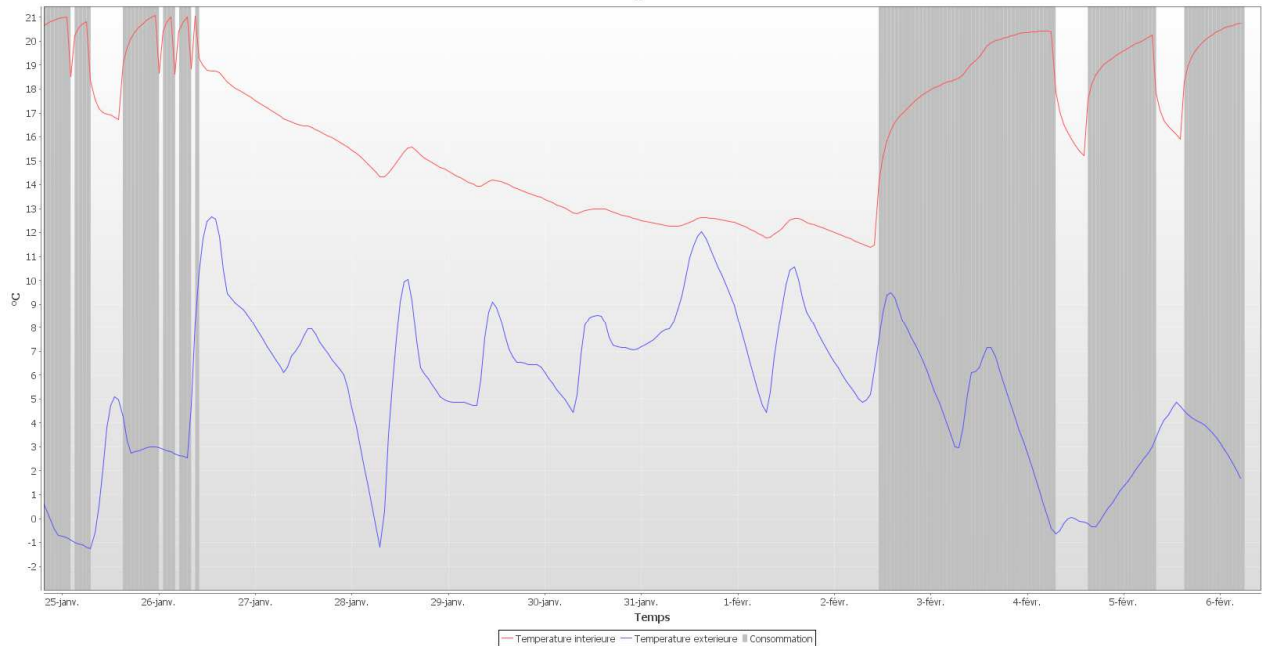


Figure 4: Activity diagram over one week



**Figure 5: Indoor and Outdoor temperature evolution over holiday period**

## References

- [1] MODELISAR – ITEA2, Functional Mock-up Interface for Co-Simulation, October 2010.
- [2] Kashif, A., Ploix, S., Dugdale, J., Le, X.H.B., Simulating the dynamics of occupant behaviour for power management in residential buildings. *Energy and Buildings* Vol. 56 (2013) p85-93.
- [3] Bourgeois, D., Reinhart, C., Macdonal, I., Adding advanced behavioural models in whole building energy simulation: A study on the total energy impact of manual and automated lighting control. *Energy and Buildings* Vol. 38.
- [4] Amouroux, E., Huriaux, T., Sempe, F., Sabouret, N., Haradji, Y., Simulating human activities to investigate household energy consumption. *Proceedings of the ICAART 2013*.
- [5] Haradji, Y., Poizat, G., Sempe, F., Human activity and social simulation. *Advances in applied human modeling and simulation*, p 416-425, 2012
- [6] Fanger, P.O., *Thermal comfort: Analysis and applications in environmental engineering*. Danish Technical Press, 1970.
- [7] Parsons, K. C., *The effects of gender, acclimation state, the opportunity to adjust clothing and physical disability on requirements for thermal comfort*. *Energy & Buildings*, vol. 34, no. 6, pp. 593–599, 2002.
- [8] REMODECE consortium, REMODECE deliverables. Can be found at <http://remodece.isr.uc.pt/>
- [9] Plessis, G., Kaemmerlen, Lindsay, A., BuildSysPro: a Modelica library for modelling buildings and energy systems. *Proceedings of the International Modelica Conference 2014*.
- [10] Pazold, M., Burhenne, S., Radon, J., Herkel, S., Antretter, F., Integration of Modelica models into an existing simulation software using FMI for Co-Simulation. *Proceedings of the International Modelica Conference 2012*.
- [11] Viel A., Strong coupling of Modelica system-level models with detailed CFD models for transient simulation of hydraulic components in their surrounding environment. *Proceedings of the International Modelica Conference 2011*.
- [12] Ptolemy consortium, Ptolemy project at <http://ptolemy.eecs.berkeley.edu/java/jfmi/>
- [13] Hindmarsh, A. C., Brown P. N., Grant, K. E., Lee S. L., Serban, R., Shumaker, D. E., Woodward, C. S., SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, Vol. 31(3), p. 363-396, 2005.
- [14] IEA Annex 60 consortium. Project webpage on <http://www.iea-annex60.org/>

# Phenomenological Li ion battery modelling in Dymola

Kotub Uddin\* and Alessandro Picarelli<sup>†</sup>

\* WMG, The University of Warwick,  
International Digital Laboratory, Coventry, CV4 7AL, United Kingdom

<sup>†</sup>Claytex Services Ltd.,  
Edmund House, Rugby Road, Leamington Spa, CV32 6EL, United Kingdom

\*k.uddin@warwick.ac.uk, <sup>†</sup>alessandro.picarelli@claytex.com

## Abstract

In this work, the structure of a modular, acausal and reconfigurable electro-thermal battery model is described. The dynamic model structure adopted for the battery cell is based on an equivalent circuit whose parameters are generated using real cycling data through an optimisation routine written in the Modelica language. A linearised one-dimensional thermal mathematical model with lumped parameters is used to simulate temperature profiles for the cell. The cell and scaled-up pack model is parameterised for a number of commercially available cells ranging a number of cell formats, sizes and chemistries. These Dymola models are validated using highly transient and aggressive real-world as well as synthetic drive cycles.

*Keywords: Lithium ion, battery, HEV, EV, PHEV, Acausal, Dymola, Modelica*

## 1 Introduction

A key enabler (or constraint) of the electrified power train is the need to store energy in a form that can be easily and robustly converted into electricity. Batteries have emerged as a preferred choice in alternative energy storage but the technology still comes with significant compromises for the customer. Many of the challenges and opportunities presented by battery technology can be traced to the li-ion cell at the heart of the battery.

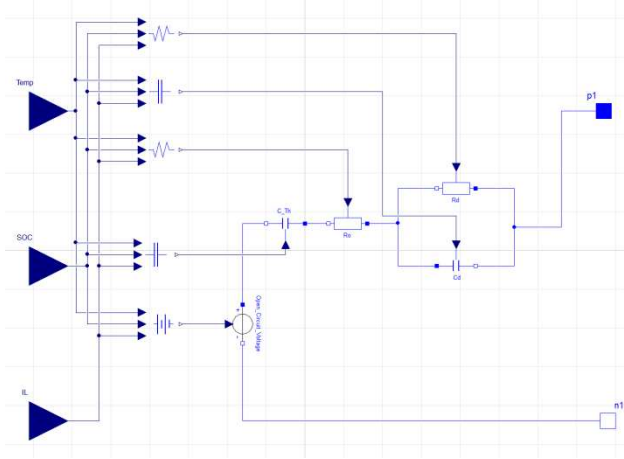
The need to accurately, rapidly and robustly model the performance of cells and their effects on the battery system and wider vehicle is of paramount importance to vehicle OEMs. While multiple modelling approaches are available for Li-ion cells, a balance is required to produce a model that has the flexibility to map the microscopic scale effects of internal cell mechanisms to the macroscopic scale of pack and vehicle dynamics in a timely and cost effective manner.

Moreover it is important for any of these battery models to be readily integrateable with an existing electrified powertrain and control simulation toolset, where an acausal simulation structure (as opposed to input-led) can be advantageous. In this paper, we extend the INEEL FreedomCar program model [1] to include temperature dependence, voltage hysteresis, self-discharge and diffusion limitation. We present a generic routine that can be used to generate model parameters based on optimisation [2]. Moreover, we utilise the capacity of Modelica to avoid assignment statements and base our model on equations so as to achieve acausality. Finally, we show an efficient method for scaling-up the cell electro-thermal model, while maintaining the ability to uniquely parameterize individual cells, to pack level without substantially compromising simulation time.

## 2 Model development

### 2.1 Cell Model

Modelling the electronic and thermal behaviour of a battery cell requires characterization of mechanisms across multiple time domains and model parameters that are dynamically interlinked. The equivalent circuit model (ECM) adopted in this work is shown in Figure 1 and consist of a parallel RC network connected in series with a capacitor, a resistor and an ideal voltage source. As shown in Figure 1, the circuit is mainly composed of three parts including an open-circuit voltage source  $V_{oc}$ , internal resistances and equivalent capacitances. The internal resistances include the ohmic resistance  $R_0$  which comprises all electronic resistances and the polarization resistance  $R_d$  which when coupled with  $C_d$  accounts for ion diffusion. The equivalent capacitance  $C_{Th}$  is used to describe the transient response during charging and discharging [1].



**Figure 1:** Depicting the equivalent circuit model of a Li-ion battery system. The circuit represents temperature, state of charge and current dependency of the circuit components; self-discharge and the hysteresis effects added with open circuit voltage.

The electrical behaviour of the ECM shown in Figure 1 is given by [1]:

$$\begin{aligned} V_L &= V_{oc} - I_L R_0 - U_{Th} - U_d \\ \frac{d}{dt} U_d + \frac{U_d}{R_d C_d} &= \frac{I_L}{C_d} \\ \frac{d}{dt} U_{Th} &= I_L \frac{dV_{oc}}{dQ} \end{aligned} \quad (1)$$

where  $V_L$  is the terminal voltage,  $I_L$  is the load current,  $U_{Th}$  is the voltage drop across the capacitor  $C_{Th}$ ,  $U_d$  is voltage drop due to polarisation effects and  $Q$  is accumulated charge; the mentioned variables are time dependent. This coupled set of equa-

tions (1) can be solved analytically and without loss of generality the solution is given by:

$$\begin{aligned} V_L &= V_{oc} - I_L R_0 \\ &\quad - \frac{1}{C_{Th}} \int I_L dt \\ &\quad - \frac{e^{-\frac{t}{R_d C_d}}}{C_d} \int e^{\frac{t}{R_d C_d}} I_L dt \end{aligned} \quad (2)$$

The last term on the right hand side arises from the RC component and is decoupled from contributions from the other components in the circuit. This suggests that from a mathematical viewpoint introducing more RC terms into the ECM will not lead to challenging parameterization algorithms, however, in so doing one must give attention to balancing the computational effort with accuracy yield. In this work we consider a single (modified) RC circuit, similar to that proposed by INEEL FreedomCar program [2], and will show that this leads to sufficient accuracy.

Equation (2) can be re-written in a closed form thus [2]:

$$\begin{aligned} V_L &= V_{oc} - I_L R_0 - \frac{1}{C_{Th}} \int_{t_0}^{t_f} I_L(t) dt - I_P(t) R_d, \\ I_{P,i} &= \left[ 1 - \frac{1 - \exp\left(-\frac{\Delta t}{\tau_P}\right)}{\frac{\Delta t}{\tau_P}} \right] I_{L,i} \\ &\quad + \left[ 1 - \frac{1 - \exp\left(-\frac{\Delta t}{\tau_P}\right)}{\frac{\Delta t}{\tau_P}} \right] \\ &\quad - \exp\left(-\frac{\Delta t}{\tau_P}\right) \Bigg] I_{P,i} \\ &\quad + \exp\left(-\frac{\Delta t}{\tau_P}\right) I_{P,i-1} \end{aligned} \quad (3)$$

which is written in matrix form:

$$\begin{pmatrix} V_{L,1} \\ V_{L,2} \\ \vdots \\ V_{L,n} \end{pmatrix} = \begin{pmatrix} 1 & I_{L,1} & Ah_{L,1} & I_{P,1} \\ 1 & I_{L,2} & Ah_{L,2} & I_{P,2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & I_{L,n} & Ah_{L,n} & I_{P,n} \end{pmatrix} \begin{pmatrix} V_{oc} \\ R_0 \\ C_{Th}^{-1} \\ R_d \end{pmatrix} \quad (4)$$



where

$$Ah_{L,i} = \sum_{k=0}^i I_{L,k} \Delta t. \quad (5)$$

Solving equation (4) via an optimisation routine in Modelica, which constitutes a part of the battery library, generates estimates for the parameters of the ECM at a fixed temperature ( $T$ ), load current and state of charge ( $SoC$ ) defined by

$$SoC = SoC(t_0) - \frac{100}{Q_{rated}} \int_{t_0}^{t_f} (I + S_D) \cdot dt \quad (6)$$

where  $Q_{rated}$  is the rated capacity of the cell (the total amount of charge that can be reversibly cycled from the cell) and  $S_D$  is the self-discharge contribution given by [3]:

$$S_D = k_0 \left( -\frac{E_A}{R_g T} \right) SoC \quad (7)$$

where the ratio of activation energy to the molar gas constant  $E_A/R_g$  is determined through observations and on timescales of a few hours can be taken to be zero. The optimisation routine runs a sweep for  $\tau$  within a range of values. The value of  $\tau$  which yields the least error when predicting  $V_L$  via equation 4 is selected for the reference data set.

The optimisation routine is repeated for various temperatures, current pulses and  $SoC$  values to produce a three dimensional map of battery parameters as a function of  $SoC$ ,  $I_L$  and  $T$ . These parameters are then fed into the equivalent circuit models.

The Hysteresis contributions  $V_h$  to cell voltage is modelled by the following first-order differential equation which we couple to open circuit voltage  $V_{oc}$  [4]:

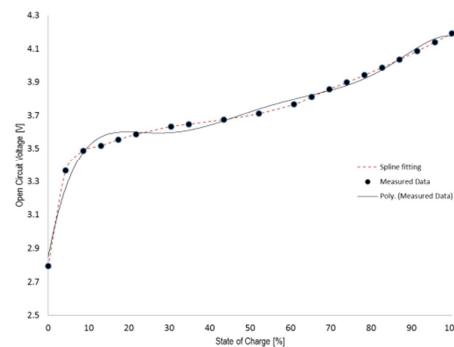
$$\frac{\partial V_h}{\partial t} = -\beta(I_L - \epsilon S_D)[V_{h,max} + \text{sign}(I)V_h] \quad (8)$$

where the constants  $\beta$  and  $\epsilon$  are to be determined and  $V_{h,max}$  is the limiting hysteresis voltage. Equation (8) is constructed such that for prolonged as well as large pulse charge currents the hysteresis voltage tends to  $V_{h,max}$  while for prolonged as well as large pulse discharge currents the hysteresis voltage tends

to  $-V_{h,max}$ . Moreover, if there is a prolonged period of zero current the hysteresis voltage tends to  $-V_{h,max}$  through the self-discharge effect.

The  $RC$  component shown in Fig 1 attributes a time constant  $\tau = R_d C_d$  to the bulk diffusional process of Li ions in the solid phase. We account for diffusion limitation, where the surface concentration of lithium may be significantly different than the average concentration contained within the active material particle, by allowing the time constant to be a function of current and assume the following simple power-law form [5]:

$$\tau = |a_0 + a_1 I_L + a_2 I_L^2 + a_3 I_L^3 + O(I_L^4)| \quad (9)$$



**Figure 2:** A comparison of  $V_{oc}$  estimation using polynomial fitting (6<sup>th</sup> order) and spline functions applied to  $V_{oc}(SoC)$  values generated through solving Eq. (4).

To generate mathematically smooth estimates for the battery parameter ( $Y = Y(SoC, I_L, T)$ ) the usual practice is to use polynomial fitting functions. This method has the disadvantage that it often does not fit the data well as in the case of Open Circuit Voltage depicted in Fig. 2. In this work we utilize spline functions to generate smooth estimates for  $Y(SoC, T)$  which avoid oscillations in interpolated values either side of outliers. We find that while both estimates follow the general trend of the data in the highly non-linear regions of the curve the accuracy of the spline function is much greater compared to a polynomial fit.

Cubic splines are preferred over lower degree splines. With first-degree splines the slope of the spline may change abruptly at the knots (i.e., data points) and for second-degree splines the discontinuity is in the second derivative which means that the curvature of the quadratic spline changes abruptly at each node. The cubic spline function  $S$  is de-

defined in the  $X$  interval  $[X_0, X_f]$  such that  $S$  is a polynomial of degree at most 3 on each subinterval  $[X_i, X_{i+1}]$  and  $S$  is continuous up to its second derivative.

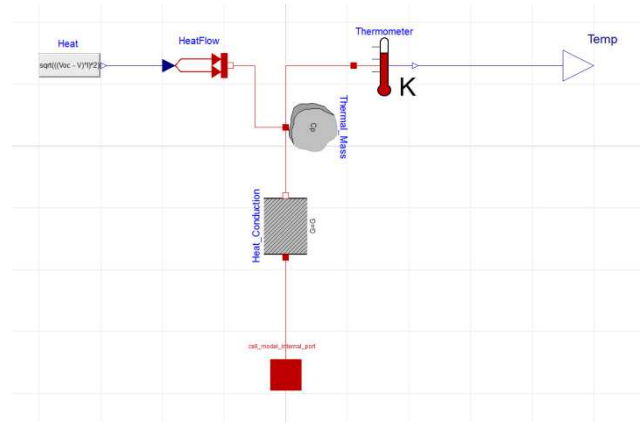
Spline interpolation is coupled with linear interpolation where there is a weak correspondence between a variable  $\{SoC, I_L, T\}$  and the model parameter  $Y = \text{ter } Y = Y(SoC, I_L, T)$ .

A further improvement in stability was achieved using Akima splines. The calculation of the derivative only relies on data from local points, hence reducing the amount of oscillations between data points in the interpolation.

Temperature is modelled as lumped value. For modelling Li-ion batteries this is convenient because of the resulting simplicity of the governing equations. Such assumptions can be suitable if temperature gradients within the cell body are negligible [5] which does not hold for most HEV applications where currents are large. It has recently been shown that a cell with a temperature gradient maintained across it has a lower impedance than one held at the theoretical average temperature [6]. Our assumption therefore will introduce some errors in voltage predictions but this, as will be demonstrated, can be negligible. The thermal model for a single cell is depicted in Figure 3 where the heat generated in [J] is given by:

$$\dot{Q} = I \left( V - V_{OC} - T_{ref} \frac{dV_{OC}}{dT} \right) \quad (9)$$

where a dot represents a time derivative and  $I(V - V_{OC})$  represents irreversible joule heating caused by Li-ion transport. The last term on the right hand side of Eq. (9) represents the reversible rate of heat generation due to entropy change. This model assumes no specific geometry for a single cell beyond a total volume with uniform temperature for some arbitrary body.



**Figure 3:** Figure depicting the 1D thermal model which is coupled to the electrical model via the heat source (equation 9).

## 2.2 Pack Model

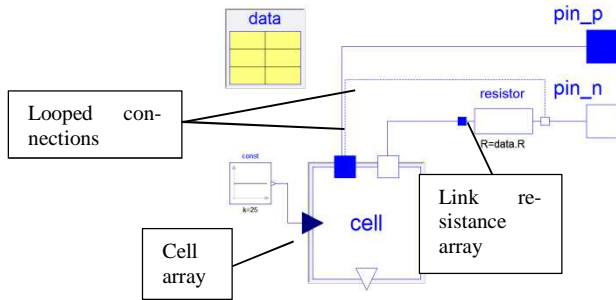
The pack model consists of an electrical network and a thermal network connecting electrical and thermal paths respectively.

### Pack electrical model

A *pack* is constructed from *modules* which are constructed from *sub-modules*. The *sub-modules* are of two types: only parallel connections and only series connections. This allows us to make effective “cells” which become the building blocks of our model. The *sub-modules* are constructed by appropriately looping electrical connections of a single cell. Each cell is then allowed to be unique, i.e., take unique values of initial *SoC*,  $C/3$  discharge capacity, internal resistance, polarisation resistance,  $V_{OC}$ , bus bar connection resistance – through a data table. Such uniqueness of cell parameters allows for various interesting studies including cell balancing, diagnostics, performance limitations, design optimisation and so forth. The *sub-modules* are either connected via parallel or series loops to construct *modules* which can be connected to construct a *pack*. The unspecified parameters of the pack electric model include bus-bar resistance and inter-module connection resistance. For studies of high frequency ripple, the inductance of bus bars can be included as a parameter; however, in what is presented here this option is deselected.

The advantages of using component arrays over separate instantiations of components is firstly: less model diagram layer space being taken up, but more importantly: the ability for the model to be scaled up

or down without having to manually redefine the architecture.



**Figure 4:** Dymola model diagram layer showing component array methodology for scalable stacks.

In the example shown in Figure 4, each element in the cell and resistance arrays (size= $n$ ) have been linked using the following notation:

```
connect(pin_p, cell[1].pin_p)

for k in 2:data.n loop
  connect(resistor[k -
    1].n, cell[k].pin_p)
end for;

connect(resistor.p, cell.pin_n)
connect(resistor[data.n].n, pin_n)
connect(const.y, cell.Temp)
```

Symbolic Manipulation is a powerful tool used to simplify the systems of equations generated for a model during compilation. The model equations are rearranged into a form where the unknowns can be calculated whilst the redundant equations are removed. Dymola like other Modelica based tools has its own version of Symbolic Manipulation which helps achieve time-effective model computation.

The simplification of the systems of equations leaves the accuracy of the model intact whilst dramatically reducing the computational effort required to solve the original model equations [7].

#### Pack thermal model

Akin to the electrical pack model we connect thermal paths between cells via *heat-ports*. Thus, employing loops we thermally connect *cells* within *sub-modules*, *sub-modules* within *modules* and *modules*

within a *pack* which is attached to a global cooling circuit. In our work we consider two cooling models:

#### Control based ideal cooling system

This simple model mimics an ideal cooling strategy. The cooling system remains inactive as long as the temperature of the cells in the pack remains below some critical temperature  $T_c$  (a design parameter). As soon as the temperature exceeds  $T_c$  the cooling system takes action: it extracts heat  $Q(T)$  [W] away from every cell in the pack uniformly. The function  $Q(T)$ , which is a function of cell temperature, can take any form and in the simplest design will be a constant (i.e., regardless of how hot the cells are, the cooling system exerts the same effort). Once the temperature is forced below  $T_c$  the cooling system will return to its idle state.

#### Fluid based Cooling Model

This model couples simple fluid dynamics with the existing electro-thermal model. This model connects every cell in the pack via their sub-modules and modules to the global cooling circuit via cooling plates. The surface area of the cooling plate is a critical feature and is a parameter of the model. The cooling system is traversed by coolant (incompressible mixture of water and ethylene glycol, the ratio of which is a parameter of the model) that is forced by an ideal pump with a mass flow rate  $m_{flow}$  that is a parameter of the model.

## 3 Validation

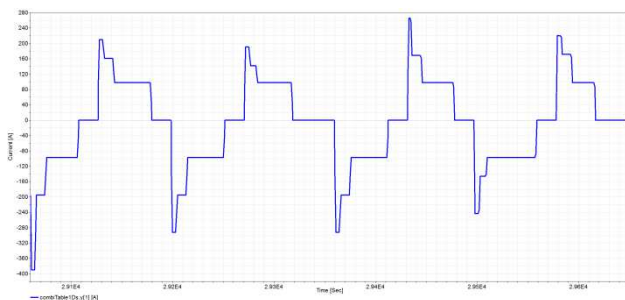
We have parameterized our model for a number of commercially available cells ranging a number of cell formats, sizes and chemistries including cells with Lithium iron phosphate (LFP), Manganese spinel (LMO) and Lithium nickel manganese cobalt (NMC) cathode materials and graphite and Lithium titanate (LT) anode materials. We find cells with the same chemical compositions have similar  $V_{OC}(SoC)$  profiles; however, internal resistance – which is influenced by factors such as electronic contact between active electrode materials and current collectors, homogeneity of the active material paste, the battery internal structure – and polarisation re-

sistance – influenced by electrolyte composition – are found to differ. It can thus be established that the manufacturing process itself will have a bearing on a batteries performance characteristics.

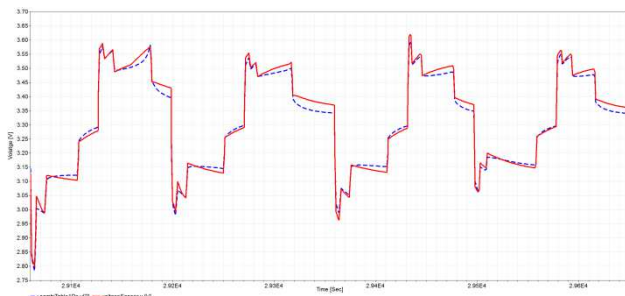
In what follows of this sub-section we present some validation results for the electro-thermal cell model. While the model has been validated for a number of commercially available cells here we present results for a 20Ah LFP pouch cell. The cells were cycled and monitored using a Bitrode MCV 16-100-5 EV/HEV Battery Cell Test System.

### 3.1 Cell level validation

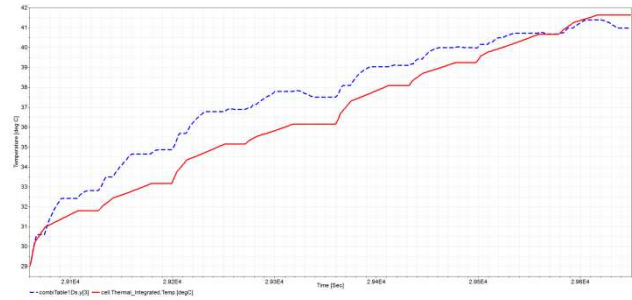
The validation of the 20Ah LFP cell utilised an aggressive artificial cycle, shown in Figure 5a, in order to test the model boundaries. The subsequent results of the electro-thermal model are shown, with laboratory test data, in Figures 5b-5c.



**Figure 5a:** Depicts a highly demanding current profile used for this validation process. The duration of this cycle is 600 seconds (excluding rest time) and the pulses range between 20C discharge and 10C charge.



**Figure 5b:** A comparison of predicted terminal voltage using the cell model (blue line) with actual test data (red line) for a 20Ah LFP cell. The maximum transient error is less than 50mV.

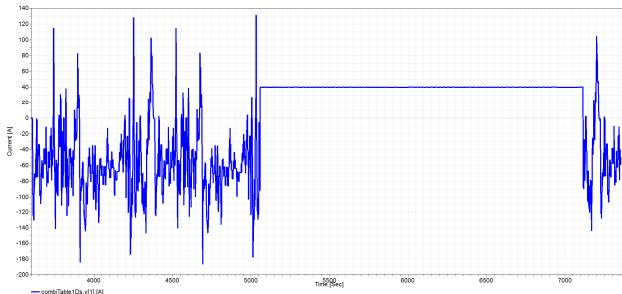


**Figure 5c:** A comparison of temperature predictions using the reduced order model (red line) with laboratory data (blue line) for an A123 20Ah LFP cell.

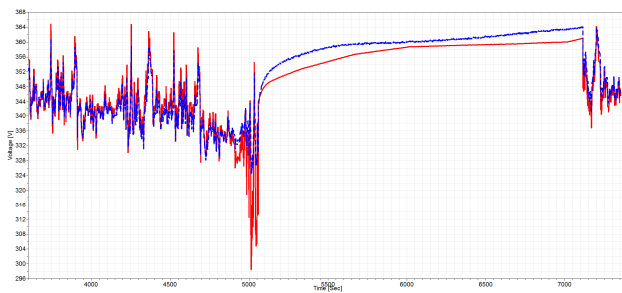
### 3.2 Pack level validation

For pack validation we use a commercial pack comprising 214 20Ah LFP cells in a 2p107 configuration. The cells are organised into 5 modules: 4 modules with 2p24s configuration and a final module with a 2p11s configuration. There is a service break that splits the pack between 2 modules and 3 modules (2p48s and 2p59s) for safety. The cells are welded to the bus bar by laser welding and the modules are connected via thick 48mm cables with a resistance of  $240 \times 10^{-8} \Omega$  and inductance of 50nH. The pack was cycled and monitored using a Bitrode FTF-500-900 EV/HEV Battery Pack Test System and temperature was recorded using t-type thermocouples connected to a pico logger.

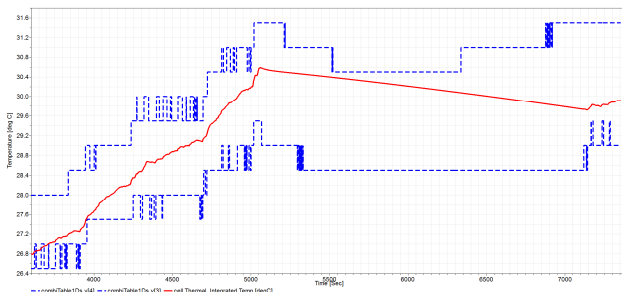
The pack was cycled with a number of PHEV cycles. Here we present results of a single charge sustaining mode cycle. It is worth mentioning that the nominal transient discrepancy between modelled and measured voltages were below 2% during most part of the simulation and in some cases peaked to 5% as shown in Figure 6b (i.e., the maximum transient discrepancy between modelled and measured voltage is less than 5%). The pack was connected to a cooling system passing liquid coolant at a rate of 27L/min thorough the cooling plates attached to the bottom of the pack (opposite end to cell tabs).



**Figure 6a:** A 63 minute current cycle consisting of three phases: The first 24 minute charge depleting phase with currents between  $-200A < I < 130A$  takes the cell from 70% SoC down to 10% SoC; this is followed by a 34 minute phase of constant 40A charge which takes the cell back up to 70% SoC; the final phase is a 5 minute charge depleting phase with similar current magnitudes.



**Figure 6b:** Shows a good fit between modelled voltage (red) and measured voltage (blue). During most of the simulation errors were below 2%, with a peak error of 5% at around the 5000 seconds.



**Figure 6c:** Compares modelled temperature (red line) versus maximum recorded cell temperature in the pack and minimum recorded temperature (blue dashed lines).

## 4 Conclusions

In this paper we present a library of models that extend to construct a coupled, dynamic, electro-thermal model of battery cells and packs. At the elementary level we utilize a single polarization equivalent circuit model (ECM) to capture the Ohmic and diffusional characteristics of a Lithium ion battery. The EC model is then developed to include effects of hysteresis, self-discharge and diffusion limitation.

A distinguishing property of this model is the inclusion of diffusion limitation effects through a current dependant time constant. This property better mimics the solid diffusional dynamics of  $Li^+$  intercalation into the active material.

Employing an optimisation routine we extract state of charge (SoC), temperature (T) and current dependent (IL) model parameters from High Pulse Power Characterization (HPPC) data. This extracted data then forms a three dimensional “look-up table” which is interpolated using Spline functions in the ranges  $0\% \leq SOC \leq 100\%$  and  $-20^\circ C \leq T \leq 65^\circ C$ .

The model (cell, module and pack) is acausal and thus utilises physical pins that mimic battery terminals. Stimuli are therefore any load acting on the battery via the tabs (as is the case in reality) and outputs are any measurements that are made across the tabs (using modelled sensors).

The cell model was validated using a number of commercially available cells. Our combined validation results showed a maximum of 50mV discrepancy between measured and tested voltages at cell level and a  $1.4^\circ C$  discrepancy between measured and tested temperature at cell level.

Our pack model is constructed from a series-parallel configuration of cells where cell-to-cell connections are modelled by a resistor. Our focus for this integrated architecture was methods of automated and unique parameterization which facilitates, for example, the study of the effects of SoC imbalance, anomalous resistances and SoC balancing mechanism.

An important characteristic of the model presented here, which is unique to Modelica code, is that it is based on equations instead of assignment statements. The main advantage is that the solution direction of equations will adapt to the data flow context in which the solution is computed. The nature of the equation based approach means that the models are acausal. This in turn means that the modeler does not have to rewrite or rearrange the system equations when using the components in different scenarios, for example forward and inverse dynamic modelling situations. More advantageously a model of a battery behaves physically like battery. So for example, if a resistor is connected to the physical pins (modelled negative and positive terminals of a battery) the battery will discharge causing a rise in temperature, fall in SOC, fall in voltage and so on.

In this work we demonstrate Dymola's ability for multi domain modelling. We find that the Mdoelica libraries including Electric, Fluid and Thermal readily facilitate the construction of coupled electro-thermal battery models. However, we also find that Dymola is lacking in some areas of numerical problem solving, particularly in solving coupled non-linear simultaneous equations as well as coupled non-linear partial differential equations. We circumvent these problems by employing regression methods for predicting battery parameters.

## References

- [1] H. He, R. Xiong, J. Fan, Evaluation of lithium-ion battery equivalent circuit models for state of charge estimation by an experimental approach, *Energies*, 4 (2011) 582-598.
- [2] I.N.E.E. Laboratory, Battery Test Manual for Plug-In Hybrid Electric Vehicles, in, Assistant Secretary for Energy Efficiency and Renewable Energy (EE), Idaho Operations Office: Idaho Falls, ID, USA, 2010., 2010.
- [3] M.W. Verbrugge, R.S. Conell, Electrochemical and thermal characterization of battery modules commensurate with electric vehicle integration, *Journal of the Electrochemical Society*, 149 (2002) A45-A53.
- [4] M. Verbrugge, E. Tate, Adaptive state of charge algorithm for nickel metal hydride batteries including hysteresis phenomena, *Journal of Power Sources*, 126 (2004) 236-249.
- [5] K. Uddin, A. Picarelli, C. Lyness, N. Taylor, Acausal electro-thermal Li-ion battery models for automotive applications, *Journal of Power Sources*, to be published (2014).
- [6] Y. Troxler, B. Wu, M. Marinescu, V. Yufit, Y. Patel, A.J. Marquis, N.P. Brandon, G.J. Offer, The effect of thermal gradients on the performance of lithium-ion batteries, *Journal of Power Sources*, 247 (2014) 1018-1025.
- [7] P.A. Fishwick, *Handbook of dynamic system modeling*, CRC Press, 2007.

# A Modelica Based Lithium Ion Battery Model

Johannes Gerl <sup>a</sup> Leonard Janczyk <sup>a</sup> Imke Krüger <sup>a</sup> Nils Modrow <sup>a</sup>

<sup>a</sup>Modelon GmbH  
Agnes-Pockels-Bogen 1  
D-80992 München

johannes.gerl@modelon.com  
imke.krueger@modelon.com

leonard.janczyk@modelon.com  
nils.modrow@modelon.com

## Abstract

*The initial integration of a large scale battery system in existing end products like cars is usually of experimental nature. So are the simulation models supporting its design process. In the following a comprehensive Modelica model is introduced for the simulative description of the physical behavior of lithium ion battery cells packs for relevant aspects and use cases. It is part of the Modelon Battery Library, a commercial Modelica library to model battery cells and packs of various types, shape and grouping.*

*Thermal behavior, electrical behavior and the impact of the degradation due to aging are considered as they influence each other.*

*The model parameters to calculate the electrical behavior are to be derived from measurements; an optimization algorithm to obtain them is integrated in the package using the Optimization Library. Functions to validate the model against these measurements are included as well.*

*As an application example the simulation of an energetic energy storage system in the model of a battery electrical vehicle is shown.*

**Keywords:** battery model; lithium-ion; behavioral modeling; electrical vehicle

## 1. Motivation

In Battery Electric Vehicles (BEV) and Hybrid Electric Vehicles (HEV) the majority of car producers focus in lithium ion based battery concepts due to their high performance density in connection with reasonably high lifetime and acceptable thermal behavior. As

these vehicles become more accepted on the market, the production numbers are supposed to increase with some positive pricing effect. It is likely that this will also make lithium ion batteries attractive for use in homes and other decentralized energy systems – especially in connection with renewable energy.

Practically all lithium ion based batteries show more or less troublesome aging behavior which reduces the lifetime to unacceptable levels, if no particular provisions are taken to avoid or reduce it. Aging appears as calendric and as cyclic effect according to the number of charging and re-charging events. The main aging effects [1] of current lithium battery systems are:

- Accumulated damages of the solid electrolyte interface (SEI) between anode and electrolyte caused by chemical reactions and physical movement due to temperature changes.
- So-called lithium plating, i.e. the deposition of metallic lithium on the anode.

Aging effects are severely influenced by the thermal load on the battery. Therefore high performance battery systems need to be kept within a certain temperature range by cooling and sometimes heating.

For whatever application, in current battery systems single cells of a certain type are arranged in stacks, modules or packages through serial and parallel alignment of the cell. Cells can have cylindrical, prismatic or so-called coffee bag shape. Apart from the electrical interconnection, the cells are integrated in some thermal design concept to cool them and reduce aging. It should be noticed that car as well as energy system manufacturers design battery systems according to the

needs of the general concept of their product. I.e. the design of the battery is not based on a unified single-type approach, but many different concepts are required to cover the large range of system requirements.



Fig. 1, Battery system of the MUTE electrical car project by TU Munich

Therefore, the battery model presented in this paper uses the cell as a base unit to be parameterized with fairly simple data sheet and empiric input. With the help of pre-defined templates, organized as shown in figure 1, the user can easily set-up a battery model as an electrical and thermal system consisting of a single cell.

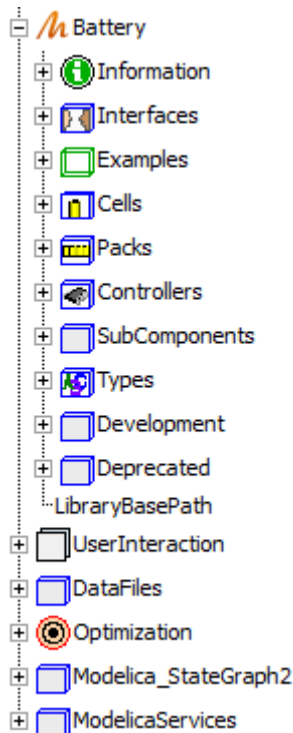


Fig. 2, Content of the Battery Library

Aging information is provided by an integrated aging system or user-defined approach.

While lithium ion battery cells are usually described by RC circuit elements, the electrochemical effects in lead-acid batteries are approximated in a separate model to take account of the specialties of this battery type.

## 2. Electrical Modelling

The main requirement for cell models used in system simulation is to provide accurate information on the macroscopic characteristics (e.g. voltage, current and state of charge) combined with reasonable computation time. In many applications these requirements are fulfilled by models using an electrical equivalent circuit.

The voltage of a battery  $U$  can be described as the difference between the open circuit voltage  $U_{OCV}$  and a number of overpotentials  $\eta_i$  caused by different electrochemical effects:

$$U = U_{OCV} - \sum \eta_i$$

These overpotentials can be modelled with electric networks. In figure 2 the voltage characteristic for the step current discharge of a NiMH cell is shown.

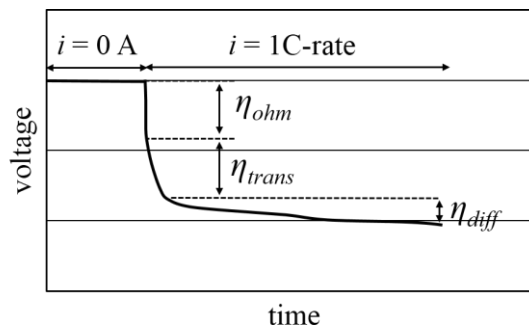


Fig. 3, Voltage characteristic of NiMH cell [1]

The overpotential is divided into an ohmic overpotential  $\eta_{ohm}$ , overpotential caused by charge transfer and the electrical double layer  $\eta_{trans}$  and overpotential due to diffusion  $\eta_{diff}$ . An electrical equivalent circuit capable of reproducing the shown voltage characteristic is shown in figure 3, whereas the dynamic behavior the overpotentials are modelled using RC-circuits.



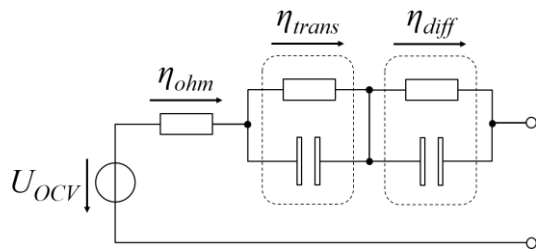


Fig. 3, Equivalent circuit

The performance of a cell is strongly depends the battery current, state of charge, temperature and other factors. To achieve good performance of the model over a wide range of conditions the consideration of these dependencies in the models of the electrical components and their parameterization is crucial.

The presented library offers equation based and table based modeling of the electrical components. As an example of equation based modeling a lithium ion cell model [2] and a lead acid cell model [3] are implemented in the library. The electrical equivalent circuits of the models contain serial resistors, RC-circuits, voltage sources representing the open circuit voltage and current sources describing the leakage current. The functions representing these elements are derived from measurement data and depend on the temperature, state of charge and current.

The table based models perform a table lookup to determine the parameters of the components in the electrical circuit. The library offers pre-defined templates for 2D and 3D interpolation. They enable a variable composition of elements in the equivalent circuit. In the 2d interpolation template the dependency of the lookup tables can easily be configured.

For the simulation of battery packs containing multiple cells, templates using discretized or scaled cell models are implemented. In the discretized pack models every cell is modeled separately. This enables the analysis of the packs' electrical behavior when unconformities of the included cells occur. As the geometric layout usually doesn't correspond to the electrical connections of the cells in the pack, a connection Matrix  $M$  is defined, that offers the possibility to connect the electric connectors of the cells in a given design.

### 3. Parameter Estimation

When modeling the electrochemical processes in a battery using a simplified approach like an electrical equivalent circuit, the choice of the circuit's components and the parameterization of these components determine the performance of the model.

A widely used approach to parameterize battery models is the generation of lookup tables from measurement data using numerical optimization algorithms ([6], [7]).

As mentioned before the battery performance is strongly dependent on numerous factors. The number of dependencies that are important for the interaction within the investigated system and the size of the range in which they need to be considered often lead to a complex optimization task.

The developed library provides a Dymola internal approach to execute parameter estimations using the commercial library Optimization developed by the German Aerospace Center (DLR) which includes several numerical optimization algorithms [10]. A template of a parameter estimation for an equivalent circuit containing a serial resistor and two RC-circuits generating 2d lookup tables is implemented. The workflow of the template is illustrated in figure 4.

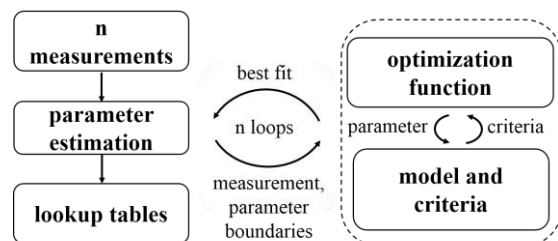


Fig. 4, Parameter estimation workflow

The inputs to the parameter estimation routine are inner resistance measurements from discharging or charging the battery with step currents. For each measurement an optimization function from the Optimization Library is called. The optimization function simulates a model that contains the equivalent circuit and computes the optimization criteria which is returned to the optimization function. The outputs of the parameter estimation are 2D lookup tables for the resistors and the capacitors in the circuit.

When computing the parameters of all components in a single estimation task the generation of plausible lookup tables is a complex

challenge [8]. To simplify this challenge and dictate e.g. which RC-circuit represents the fast dynamics and avoid a switch of assignment during the estimation task, the boundary  $b$  for each parameter can be set by

$$b = k_1 + k_2 \cdot \Delta R^{k_3}$$

$\Delta R$  is the increase of the inner resistance during the measurement and  $k_i$  are constants defining the boundary. This rather simple method showed acceptable results estimating current and state of charge dependent tables for a NiMH cell.

#### 4. Thermal Model

In order to determine the influence of varying temperatures on electrical and aging behavior a thermal model of the cell and its surrounding environment is required. Heat inside the cell is generated mainly due to Joule effects, the chemical reactions are only weakly exothermic or even endothermic. Thus the generated heat corresponds to the power loss calculated in the resistors of the equivalent electric circuit which are therefore connected to the thermal model.

##### Cell

The thermal model uses a template/interface structure with a replaceable thermal model such that the discretization level can be adapted by the user. All models are based on a finite volume approach, using heat resistors and thermal capacities. The user can choose between 0D and 1D models, further models can be added easily

Conditional heat ports at the top, side and bottom of the cell reduce the complexity without reducing the flexibility of the thermal management design.

Equations for the calculation of thermal parameters are provided for cylindrical and prismatic forms. Material records for the most common materials are also included.

##### Packs

In addition to the cell, the thermal model of the pack might consider housing and in case of the discretized pack a filling material in between the cells. Simple heat transfer models for convection and radiation are also included.

For the scaled models, the heat flow of a single instance of the cell is multiplied with total number of cells. Effects such as heat conduction in-between the cells can only be considered in the discretized pack models with several instances of the cell model. Heat transfer via pins can also be modelled; the connections between the pins use the same connection matrix as the electric part.

A two-dimensional heatport simplifies the icon of the housing, Cells, filling, and the exterior heat ports can easily be connected.

The temperature of the pack can be monitored with a provided controller model. Based on given limits. Boolean signals for activation of heating or cooling are emitted.

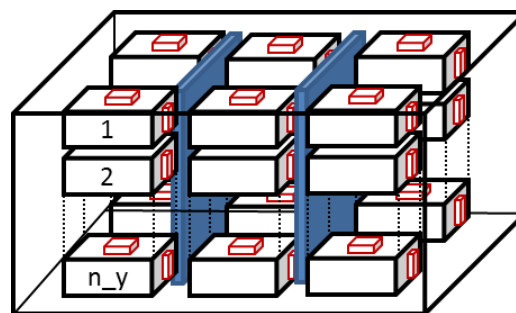


Fig. 6, Structure of thermal model of a discretized battery pack

#### 5. Aging Model

The capacity as well as the behavior of a cell change with age and cycle numbers of the cell. To account for the effects of the most important factors temperature, current rate and SOC, a flexible aging model based on the StateGraph library has been implemented.

The aging factor  $\varphi$  denotes the ratio between the current value and the value at  $t=0$ :

$$\varphi_A = \frac{A(t)}{A(t=0)}$$

Using this definition, the actual value can be determined just by multiplying with  $\varphi$ . Resistor and capacitor models have a conditional input for the aging factor that can be activated in the parameter dialog.

The flow chart in figure 6 shows the signal flow structure in the aging model. The cycle detector detects the end of a cycle and triggers the calculation of the aging factors in the cyclic and calendric aging models. Mean values

for temperature, depth-of-discharge (DOD), voltage and current are calculated for the previous cycle as the aging models are all based on continuous boundary conditions. The aging factors are discrete values, thus they are constant during one cycle until the next calculation is triggered. Therefore, the aging factors represent the age of the cell at the beginning of the next active cycle. Start value for all aging factors is set to 1.

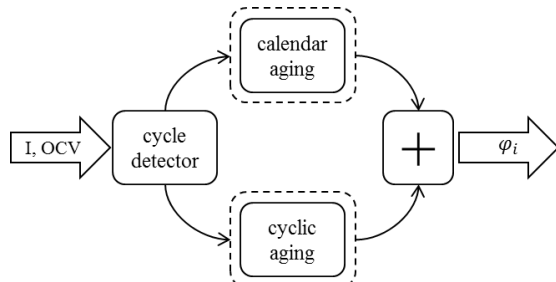


Fig. 7, Flow chart of the aging model mechanism

Both aging models use a semi-empirical approach to determine the aging factor based on recent publications ([4], [5]). The aging models are replaceable and can be switched on/off individually. New aging models can easily be added by using the provided interface.

The aging of cells during storage (calendar aging) is mainly caused by electrolyte decomposition and the growth of the solid electrolyte interface. Ecker et al [5] describe this process with a square root dependency on time. For voltage and temperature, an exponential approach is chosen. The implemented model is based on extensive measurements on 30 NMC cells stored at different SOCs and temperatures. The semi-empiric approach allows the user to adopt the parameters to his data even with a low number of measurements. The calendar model calculates aging factors for the cell capacity, the serial resistance and the parameters of the first RC-circuit. Thus, the degradation of capacity as well as the loss of power and changes in the dynamic behavior due to calendar aging can be shown.

It is supposed that the loss of active lithium due to anode degradation is the cause of capacity loss due to cycling of the cell [5]. Wang et al performed measurements with varying time, temperature, depth of discharge and discharge rate. They developed a generalist model for cyclic aging that can be adapted to different Li-Ion chemistries as long as the aging mechanisms are also based on diffusion processes. By using the energy throughput of

the cell as input of the aging factor calculation instead of time, the equation becomes independent of the charge rate. As the experiments showed little influence, the SOC of the cycled cell is neglected.

Figure 7 shows the aging factors for a cell cycled with a constant current rate between 0.5 and 0.55 SOC at 20°C. The aging factors for capacities decrease, those for the resistance increase, both reducing the capacity as well as the power of the cell.

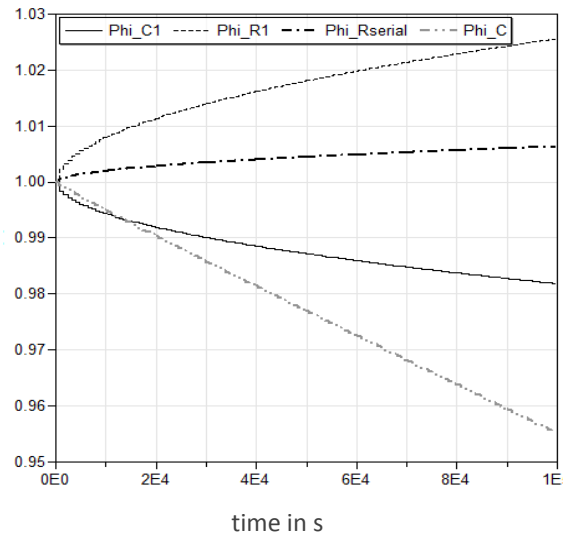


Fig. 8, Aging factors for cycling a cell between 0.5 and 0.55 SOC with 0.5C-Rate at 20°C

## 6. Application Example

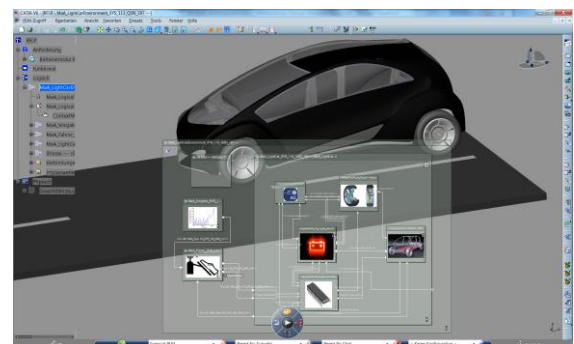


Fig. 9, Battery Lib within a Catia V6 Systems model of an e-car

In order to demonstrate the application of the Battery Library within a vehicle environment, an example project has been created based on the “Light Car” – a battery electric vehicle concept designed by the company EDAG. The development methods applied were chosen to

replicate the real-world methods as closely as possible. The partners participating in the project are the companies EDAG (to represent the car development competence), Transcat (to represent Catia V6) and Modelon (to represent system simulation with Modelica). Dimensioning the battery in terms of vehicle autonomy and its aging behavior are also in the scope of the project as well as the first time application of state-of-the-art development tools. In order to realize typical driving scenarios and test cycles like NEDC the system simulation model consists of a longitudinal dynamic vehicle model, the driving resistances and a driver. The focus of the model lies on the battery system, including its electrical, thermal and aging behavior and the battery controller.

In current e-car projects, the battery cells are supplied by cell manufacturers, but combined to a battery system at the OEM car producer. The control of the battery's primary states such as current, cell temperature, state of charge and state of health has to be in line with the entire car concept and is therefore OEM work, too. The key design factors of the battery system are the arrangement of the cells in stacks and packs under maximum utilization of their potential in terms of performance and duration of life. In this context, due to the very high influence of the temperature on the aging, certain limits have to be kept during all conditions of operation. In the presented project, the cooling design is based on air flow. For a maximum precision of the calculations, the 1D system simulation results from the Modelica / Catia V6 Systems environment have been verified using a finite element simulation in the tool Simulia.

A typical simulation result in terms of a cell temperature is shown in Fig. 9.

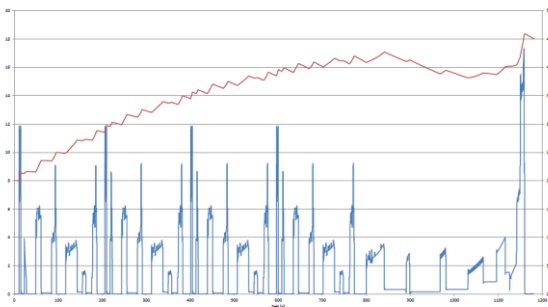


Fig. 10, Temperature (red) and heat performance (blue) of a battery cell within the NEDC drive cycle

## 7. Summary and Outlook

The battery model of Modelon's Battery Library for Lithium Ion cells has been described in its structure, functionality and employment.

The calculation of the electrical behavior by equivalent circuit models with table based and equation based approaches has been shown as well as its parameterization function based on the Optimization Library. The thermal model has been described for single battery cells and battery packs. The estimation of degradation due to cell aging has been modelled in different ways, calendric aging and cyclical aging. An example for the integration of the battery model in a vehicle simulation of an electrical car has been described in the "Light Car" project.

The Battery Library was designed with the intention to be coupled with other system models in Modelica-based or other simulation frameworks. As the battery model features all necessary interfaces, the code of a battery management system in a signal flow simulation environment can be attached as an FMU. Vice versa, it is possible to use this physical battery model for the prediction of the thermal-electrical behavior as well as aging on a battery management system or for a "model in the loop" approach.

The ongoing development of the Battery Library is heading towards electrochemical modelling of the aging behavior by the implementation of a "Dual-Foil-Model" [9] and the advancement in the thermal modelling of large electrical energy storage systems comprising several battery packs. Introduction of models for capacitors of the "super capacitor" type is also planned.

## References

- [1] Andreas Jossen und Wolfgang Weydanz, *Moderne Akkumulatoren richtig einsetzen*. Book (German), Reichhardt Verlag, 2006.
- [2] Min Chen and Gabriel A. Rincón Mora. *Accurate Electrical Battery Model Capable of Predicting Runtime and I–V Performance*. In: IEEE Transactions on Energy Conversion, Vol. 21, No. 2, June 2006.
- [3] Massimo Ceraolo. *New Dynamical Models of Lead–Acid Batteries*. In: IEEE Transactions on Power Systems, Vol. 15, No. 4, November 2000.
- [4] J.Wang et al. *Cycle-life model for graphite-LiFePO4 cells*. In: Journal of Power Sources 196, pp. 3942 - 3948, 2011.
- [5] M. Ecker et al. *Development of a lifetime prediction model for lithium-ion batteries based on extended accelerated aging test data*. In: Journal of Power Sources, Vol. 215, pp. 248 - 257, 2012.
- [6] Robyn A. Jackey, Gregory L. Plett and Martin J. Klein. *Parameterization of a Battery Simulation Model Using Numerical Optimization Methods*. In: SAE Technical Paper 2009-01-1381, 2009.
- [7] Tarum Huria, Massimo Ceraolo, Javier Gazzarri and Robyn Jackey. *High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells*. In: Electric Vehicle Conference (IEVC), 2012 IEEE International, March 2012.
- [8] Robyn Jackey, Michael Saginaw, Pravesh Sanghvi, Javier Gazzarri, Tarum Huria, and Massimo Ceraolo. *Battery Model Parameter Estimation Using a Layered Technique: An Example Using a Lithium Iron Phosphate Cell*. In: SAE International Technical Paper 2013-01-1547, 2013.
- [9] M. Doyle, T. F. Fuller, and J. Newman. *Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell*. Journal of the Electrochemical Society, vol. 140, no. 6, pp. 1526 – 1533, 1993.
- [10] Andreas Pfeiffer. *Optimization Library for Interactive Multi-Criteria Optimization Tasks*. In: 9<sup>th</sup> International Modelica Conference, September 2012.



# Behavioral Modeling of Power Semiconductors in Modelica

Patrick Denz  
Vorarlberg Univ. of Appl. Sc.  
Dornbirn/Austria  
[Patrick.Denz@omicon.at](mailto:Patrick.Denz@omicon.at)

Thomas Schmitt  
Modelon GmbH  
Munich/Germany  
[Thomas.Schmitt@modelon.com](mailto:Thomas.Schmitt@modelon.com)

Markus Andres  
Modelon GmbH  
Munich/Germany  
[Markus.Andres@modelon.com](mailto:Markus.Andres@modelon.com)

## Abstract

*This paper introduces behavioral (macro) models of power semiconductors, i.e. diodes, MOSFETs and IGBTs, being part of a library for simulating power electronics utilized, e.g. in electrified powertrains of either hybrid electric vehicles (HEV) or purely battery electric vehicles (BEV). The models consider static, dynamic (switching mode) and thermal effects and in most cases can be fully parameterized solely on the basis of characteristic curves and parameters specified in datasheets. The main purpose of behavioral models is an accurate representation of the semiconductor signals to, e.g. calculate the overall losses. The MOSFET models are verified in simulations with various test circuits and are validated with measurement data provided by a company developing electric drive systems. Furthermore, the arising numerical problems are discussed and possible solutions are provided on how to modify the models in order to use them in e.g. system simulation.*

**Keywords:** *power electronics, power semiconductors, macro modeling, behavioral modeling, numerical performance*

## 1 Introduction

In practice, models of electric powertrains - consisting of at least a high-voltage battery, an inverter and an electric machine - have to answer questions regarding lifetime, maximum driving range, temperature development or overall efficiency. One of the most important and challenging tasks is to provide models that can be parameterized easily and simulated fast and robust, i.e. numerically stable.

Models of power electronic components are available in different degrees of complexity in freely- and commercially distributed libraries. When needed for industrial use, ideal models are often not accurate

enough, whereas physical ones cannot be parameterized with standard datasheets and are rather suited for the field of research. Moreover, such models are often solely available in a specific level of detail which either results in unreasonable simulation times in case of very detailed models or a lack of information in case of e.g. ideal models. In [6] different modeling techniques of the switch models available in the Modelica Standard Library (MSL) are discussed.

A trade-off between these ideal and micro modeling techniques is a macro modeling approach called *behavioral modeling*, which was first introduced for power semiconductors in [8] and is further developed at Modelon GmbH. The idea of this technique is to describe the component's behavior via characteristic curves and parameters provided in datasheets. Thus, on the one hand behavioral models of MOSFETs can be parameterized solely on the basis of datasheets and on the other hand, the models behave as specified by the manufacturer under nominal conditions. In case of IGBTs due to their internal semiconductor structure, the occurring tail current has to be measured in advance. Moreover, in trench/field-stop IGBTs due to the additional field-stop layer added to the semiconductor structure the model developed in [8] is not valid anymore and has to be modified.

Since behavioral models provide detailed switching slopes, the simulation performance is totally unacceptable if such models are used to simulate, e.g. the driving range of an electric vehicle. Hence, some possibilities are discussed on how to use such models to derive table based models that store solely the information needed for a specific simulation task.

## 2 Behavioral Modeling of Diodes

In the Modelica Standard Library (MSL) several different models of diodes are available (refer to [6]). The ideal diodes are modeled using parameterized curve descriptions, whereas the physical ones are described

by the well-known Shockley equation. These models are perfectly suited for, e.g. circuit analysis. However, if conduction- and switching losses of a specific diode are of interest none of these models can be used since they neither include the appropriate equations describing the transient (switching) behavior nor cannot be parameterized using datasheets. Behavioral models provide the means to make use of datasheet values to model the static- and dynamic behavior.

### 2.1 Static Model

In the simplest case, the static model of a diode can be described as depicted in Figure 1. The current through the diode is measured using a current sensor and serves as input signal to a table which stores the forward characteristic  $V_f = f(I_f)$  specified in the datasheet. The corresponding forward voltage drop is fed into a signal-controlled voltage source. The additional ideal reverse blocking diode ensures that the current solely flows in forward direction.

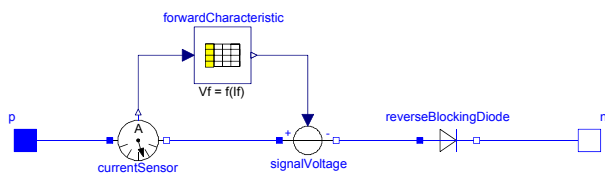


Figure 1: Static model of a diode:  $V_f = f(I_f)$

Usually datasheets provide the diode's forward characteristic not only as a function of the forward current  $I_f$  but also of the temperature  $T$ . Hence, the model has to be modified as depicted in Figure 2.

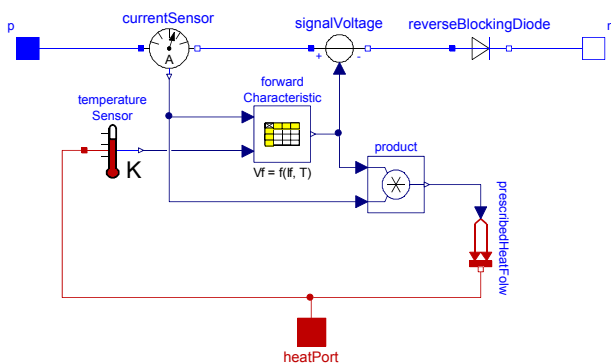


Figure 2: Static model of a diode:  $V_f = f(I_f, T)$

The losses generated while the diode is conducting are nothing else but a thermal heat flow which will result in a certain temperature depending on the thermal network connected to the heat port.

### 2.2 Dynamic Model (Reverse Recovery Effect)

The dynamics of a diode occur due to its junction- and diffusion-capacitance. An approach of modeling these capacitances is given in [8]. The main problem is that the capacitance values depend on parameters that are not available in standard datasheets. As the dynamics of a diode in terms of switching losses are mainly dependent on the *reverse recovery effect*, an approach has been followed which was published in [3].

## 3 Behavioral Modeling of Power-MOSFETs

In order to understand the behavioral model of a power-MOSFET its structure will be discussed briefly. After introducing the models its modes of operation will be verified using different test circuits. Finally, the model is validated with measurement results provided by a company.

### 3.1 Power-MOSFET Structure and its Modes of Operation

By means of the MOSFET's semiconductor structure, the different modes of operation shall now be analyzed. Based on that, a static model can be developed. In Figure 3, the structure of a vertical power-MOSFET is illustrated.

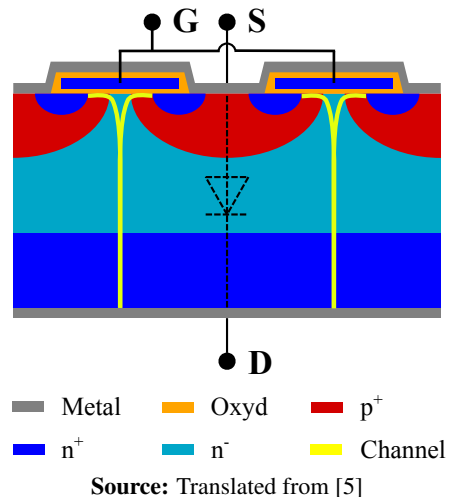


Figure 3: Structure of a power-MOSFET

It is first supposed that a positive gate-source voltage is applied, i.e. a conducting channel between drain and source arises. Now, the MOSFET operates in on-state. Thus, if a positive drain-source voltage is applied (on-state forward conduction mode),



current flows from drain to source, which represents the MOSFET's first quadrant operation. Since the conducting channel is not a pn-junction but more of a voltage-controlled resistance, current can also flow from source to drain if a negative drain-source voltage is applied (on-state reverse conduction). This was not considered in the model developed in [8] and will be added to the model demonstrated in this paper. In the modified version of the MOSFET, the characteristic curves in the first quadrant are mirrored into the third quadrant. The only difference in third quadrant application is an additional parasitic diode - the *body diode* - between drain and source. If the current in reverse direction is high enough to cause a voltage drop equal to the body diode's threshold voltage, the body diode starts conducting and provides an additional current path. A more detailed description of these reverse conduction modes can be found in [9, p. 61 ff.]. In order to analyze the off-state modes, the gate-source voltage is supposed to be zero meaning that the conduction channel does not arise. Still, the body diode can conduct current in reverse direction. In forward direction, no current can flow. In summary, the discussed modes of operation are listed below.

- Mode 1:* on-state forward conduction
- Mode 2:* on-state reverse conduction
- Mode 3:* on-state reverse conduction with body diode forward biased
- Mode 4:* off-state reverse conduction
- Mode 5:* off-state forward blocking

### 3.2 Power-MOSFET Model

Based on the modes of operation, a behavioral model of the power-MOSFET is developed in the following. First, a static model is derived, which afterwards is extended to cover dynamics and temperature dependency.

#### 3.2.1 Static Model

In order to describe the static behavior of a MOSFET, the modes of operation discussed before have to be realized in a model structure. As depicted in Figure 4, the transfer behavior and therefore the MOS-structure is modeled by a voltage sensor measuring the applied gate-source voltage. This voltage signal is the input to a table which stores the transfer characteristic  $I_d = f(V_{gs})$  specified in the datasheet. The table's output is the maximum current, which can flow due to the

applied gate-source voltage. This current value is the input to the signal-controlled current sources. Since the current that flows through the component is determined by the external load circuit, e.g. by an inductive load, each current source has an ideal diode in parallel ensuring that the current that is not drawn by the load can free-wheel through these diodes. In Figure 4 the current paths in the different modes of operation are marked with arrows. In mode 1, the current flows through the resistor  $R_{onFw}$ , which represents the conducting channel. Another ideal diode is connected in series which ensures that the current solely can flow from drain to source. In the second mode, the current flows through the resistor  $R_{onBw}$  and again, an ideal reverse blocking diode ensures that the current flows in the right direction. The body diode provides the current path for the third and the fourth mode. If the MOSFET is in on-state, the current will split between the reverse leg and the body diode as soon as the body diode's threshold voltage  $V_f$  is reached. In off-state, the MOSFET is a simple diode conducting in reverse direction and blocking in forward direction.

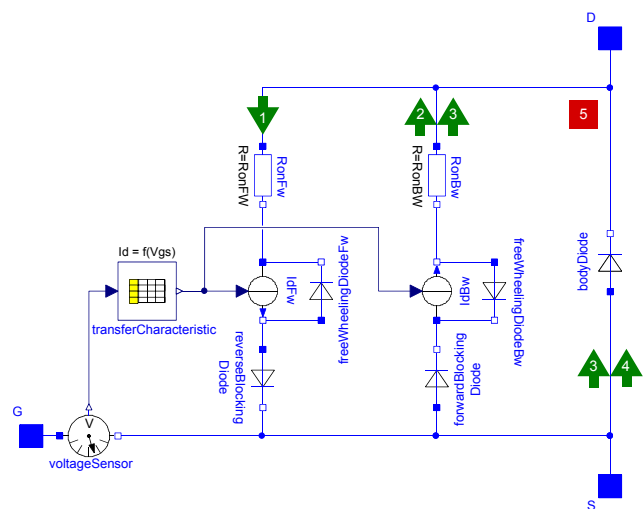


Figure 4: Static model of a power-MOSFET

#### 3.2.2 Dynamic Model

For solving a demanding simulation task, a static model is often not sufficient enough. Especially when switching losses are of interest, a dynamic model is indispensable. The static model shall now be extended to introduce the component's dynamics, which are mainly an effect of capacitances between the MOSFET's connections. As the manufacturers do not provide these directly but in form of the input capacitance  $C_{iss}$ , the output capacitance  $C_{oss}$  and the reverse trans-

fer capacitance  $C_{rss}$ , the following conversions have to be made to gain the effective capacitances between gate, drain and source:

$$C_{ds} = C_{iss} - C_{rss}$$

$$C_{gd} = C_{rss}$$

$$C_{gs} = C_{oss} - C_{rss}$$

These equations are computed in the *capTable* block, which can be seen in the dynamic model in Figure 5. The characteristic curves  $C_{iss} = f(V_{gs})$ ,  $C_{oss} = f(V_{gs})$  and  $C_{rss} = f(V_{gs})$ , specified in the datasheet are stored inside tables. The computed values are the inputs to the signal-controlled capacitors, which can be seen in the model. The MOS-structure is modeled with an RC-circuit. The gate-source capacitor is charged through the internal gate resistor and the gate-source voltage is the voltage across the capacitor.

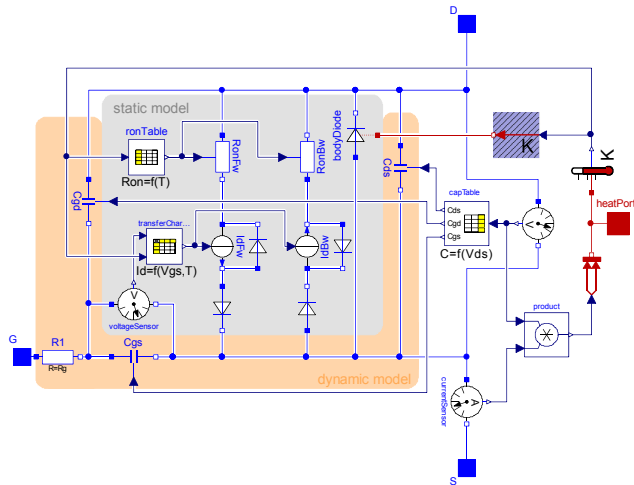


Figure 5: Dynamic model of a power-MOSFET

### 3.2.3 Temperature Dependency

Besides dynamics, temperature dependency is introduced to the model in Figure 5. Thus, in the transfer characteristic table a second input was introduced, i.e.  $I_d = f(V_{gs}, T)$ . The constant resistors have been replaced and are controlled by the characteristic curve  $r_{on} = f(T)$ . The body diode model was extended such that it considers temperature in the diode's forward characteristic table  $V_f = f(I_f, T)$  and that switching losses can be computed, i.e. the reverse recovery effect was modeled according to [3]. The total power losses are computed by multiplying the drain-source current and voltage and forwarded as heat flow to the heat port. This enables the use of a thermal network to compute the device's junction temperature.

### 3.3 MOSFET Model Verification

In order to verify the behavior of the static model in Figure 4, two test circuits have been created. For verifying modes 1, 2, 4 and 5, the circuit in Figure 6 is used. In order to test the on-state modes, the gate-source voltage is set to 10V; for testing the off-state modes it is set to 0V. The constant supply voltage source is either positive or negative depending on whether a forward conduction mode or a reverse conduction mode is tested. With the chosen load resistance, a current of approximately 100A will be drawn.

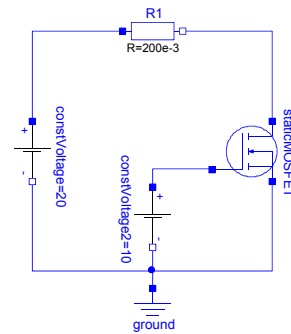


Figure 6: Test circuit for modes 1, 2, 4 and 5

Figure 7 shows the simulation results of the test circuit. It can be seen that in mode 1, the total current flows in forward direction through the resistor *RonFw*. In mode 2, the input voltage is set negative and therefore the total current flows in the backward branch through the resistor *RonBw*. In the diagram of the fourth mode, it is shown that the current is neither flowing through *RonFw* nor through *RonBw* but through the body diode. The last diagram proves that no current at all flows in mode 5.

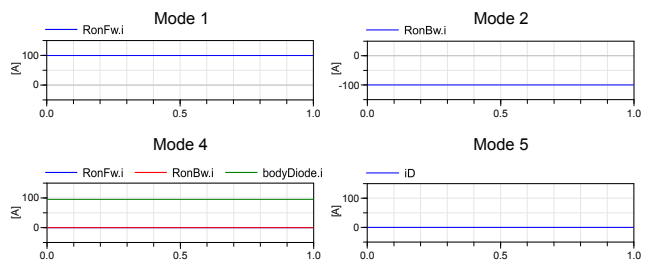


Figure 7: Simulation results for modes 1, 2, 4 and 5

To test the third static mode, the test circuit has been adapted in the way it is shown in Figure 8. The constant supply voltage source has been replaced by a signal-controlled voltage source with a ramp input. This causes the input voltage and therefore the current to increase linearly.

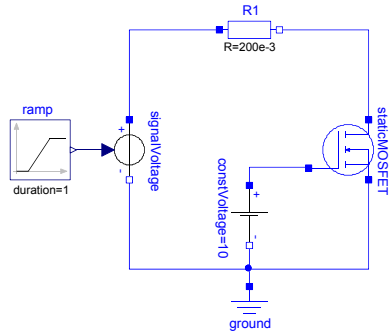


Figure 8: Test circuit for mode 3

Figure 9 shows the simulation result. The input voltage decreases from  $-20V$  to  $-50V$ . At the beginning, the current flows in the backward branch through the resistor  $R_{onBw}$ . At an input voltage level of about  $-31V$ , the current through this resistor produces a voltage drop equal to the diode's threshold voltage. Hence, the diode starts conducting and builds a current divider together with  $R_{onBw}$ . The total current is then divided into the two branches. The green curve represents the sum of the two branch currents, which is the total linearly decreasing current.

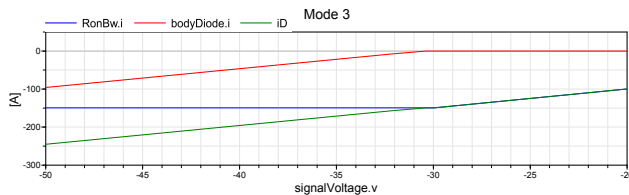


Figure 9: Simulation result for mode 3

The test circuit in Figure 10 is used to verify the MOSFET's dynamic behavior. The device is alternately turned on and off by the pulse voltage pattern applied between gate and source. A constant junction temperature is assumed and applied externally.

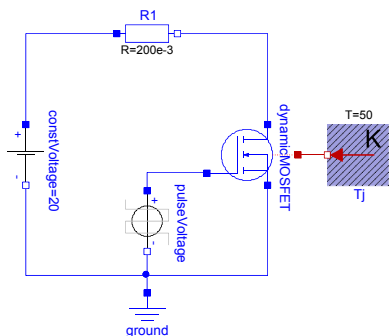


Figure 10: Test circuit dynamic behavior

Figure 11 depicts the simulation results. In the up-

per diagram, one can see the drain current, the drain-source voltage and the power losses of the MOSFET. One can see the conduction losses and the turn-on and turn-off peaks whenever the device is switched. In the lower diagram, the switch-on behavior is shown more detailed.

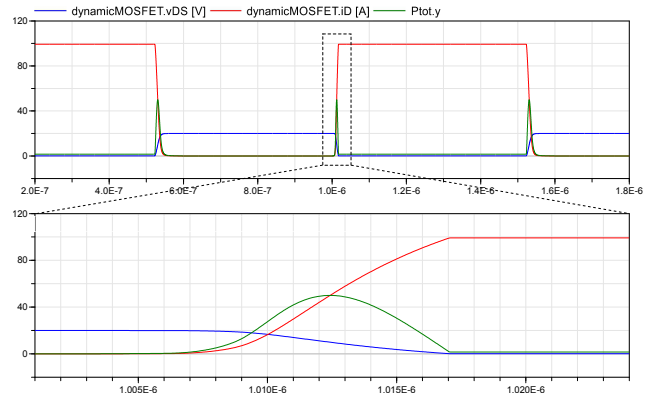


Figure 11: Switching waveforms and power losses

### 3.4 MOSFET Model Validation

The developed power-MOSFET model was validated with measurement data provided by a company producing electric drive systems. The data was obtained by measurements of a three-phase inverter of one of the company's drive systems. The circuit diagram of this inverter is shown in Figure 12. The device under test (DUT) is an Infineon IPB180N06S4-H1 power-MOSFET. Per phase leg, there are three high- and three low-side MOSFETs to be able to drive the desired load current. The company provided two space-vector modulation switching patterns for two different operating points of the electric machine and the corresponding power losses occurring per MOSFET. The data can be seen in Table 1:

Table 1: Operating points and loss power

	RPM	Torque	Power
Operating point 1	3700rpm	3Nm	1.1W
Operating point 2	700rpm	3Nm	0.9W

Since a synchronous machine with three pole pairs is connected to the inverter, a mechanical revolution speed of  $3700rpm$  results in an electrical frequency of  $185Hz$ . In order to simulate the MOSFET's total power losses correctly, they have to be averaged over this frequency. The resistive-inductive load representing the electric machine, was parameterized in such a way that the measured current for the appropriate operating point is drawn. The three phase currents and the

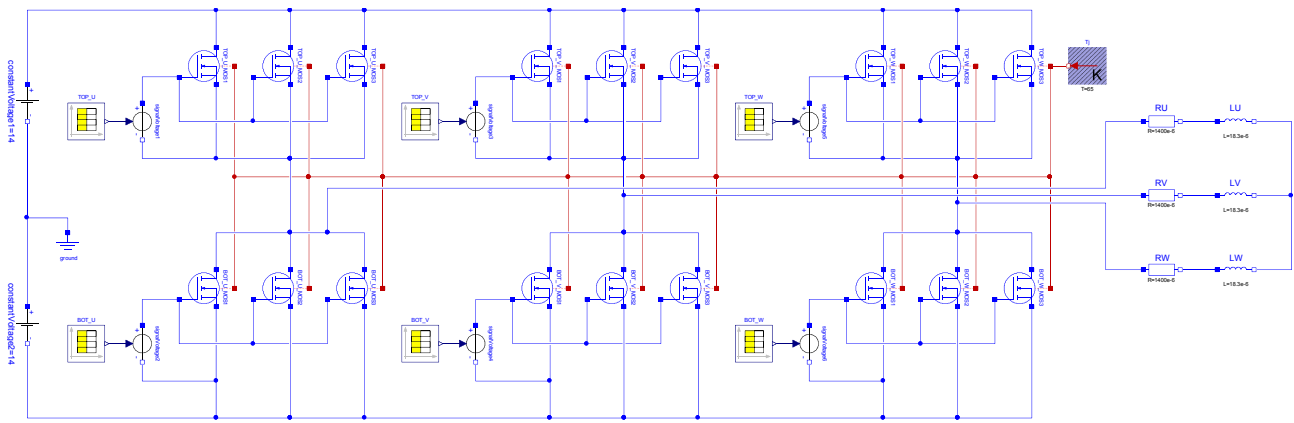


Figure 12: Power-MOSFET based three-phase inverter for driving an electric machine

average power losses per MOSFET are shown in Figure 13. The two curves in the lower diagram represent the power losses of a high- and a low-side MOSFET of the U-phase. The first average value can be computed after one period. It can be seen that once the U-phase current (blue signal in the upper plot) reaches steady-state, the average power losses over one period settle down at 1.12W.

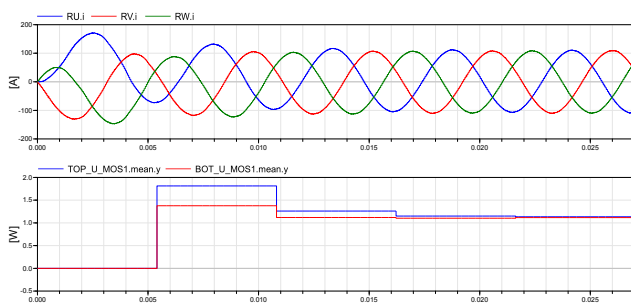


Figure 13: Simulation results of operating point 1

Figure 14 depicts the simulation results of the second operating point. At 700rpm, the inverter’s output frequency is 35Hz. Again, one can see the average power losses per MOSFET settle down in steady-state at 0.91W.

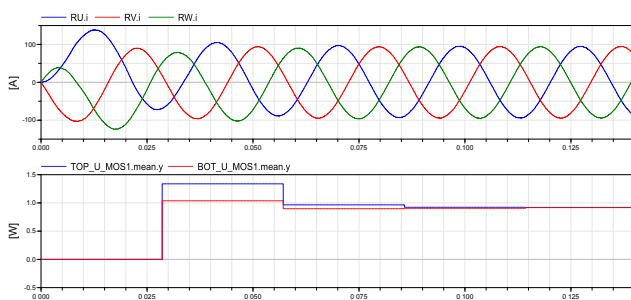


Figure 14: Simulation results of operating point 2

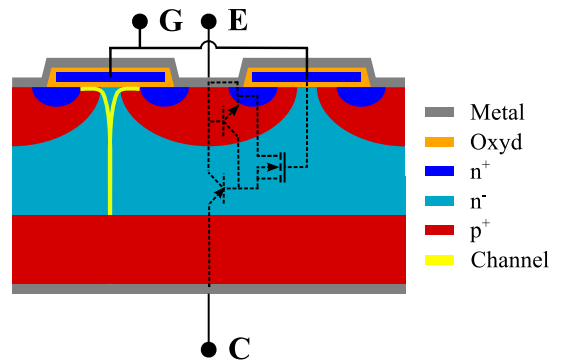
The relative error between the measured data and the simulation result lies within 2% in both operating points. These results speak especially for the behavioral modeling technique as well as for the accuracy of the provided information in the datasheets.

## 4 Behavioral Modeling of IGBTs

### 4.1 IGBT Structure and its Modes of Operation

Concerning the IGBT technology, two different design principles are available, NPT- (non-punch through) and PT- (punch through) IGBTs. The basic structure of an NPT-IGBT is illustrated in Figure 15.

Contrary to power-MOSFETs, NPT-IGBTs have an additional p+ doped layer between emitter and collector. This means that the forward characteristic of such a device does not behave like a classic resistance but more like a pn-junction. This pn-junction basically determines the IGBT’s behavior in its forward and reverse mode.



Source: Translated from [5]

Figure 15: Structure of a non-punch through IGBT

If an additional n+ doped layer is introduced between the p+ doped layer at the bottom and the n-doped layer, a PT structure is realized. Due to the additional n+ doped layer the shape of the electric field gets steeper which in turn allows for a smaller n-doped layer, i.e. reduces the chip size (refer to [7] for a more detailed description).

## 4.2 IGBT Models

A static and a dynamic behavioral model of an IGBT is introduced within this section, whereas the dynamic model is different for the NPT- and the PT-IGBT. The verification of the models is similar to the one of the MOSFET and is therefore not discussed in the paper. The validation of the models is currently under development.

### 4.2.1 Static Model

Figure 16 shows the static model of an IGBT. The output characteristic is modeled the same way as the diode's forward characteristic illustrated in Figure 1, i.e. the collector-emitter current is measured with a current sensor and fed into the table providing the IGBT's output characteristic  $V_{CE} = f(I_C)$ . The table's output is then fed into the signal-controlled voltage source. The transfer characteristic is modeled analogously to the MOSFET: The gate-emitter voltage is measured and the maximum current - provided by the transfer characteristic curve which is stored in a table - flows through the signal-controlled current source, i.e. it again behaves like a voltage controlled current source. As for the MOSFET model, a free-wheeling diode provides a path for the current which is supplied too much and a blocking diode ensures that the IGBT is not conducting in reverse direction.

### 4.2.2 Dynamic NPT- and PT-IGBT Model

To introduce dynamics to the model, three capacitors are introduced to the static model discussed before. The gate-emitter capacitor can be approximated as constant over the collector-emitter voltage (refer to Equation 1). The gate-collector capacitor - also known as *Miller-Capacitance* - strongly depends on the collector emitter voltage and can be approximated with Equation 2 [1, p. 15]. The input capacitor  $C_{iss}$  and the reverse transfer capacitor  $C_{res}$  can be found in the datasheet.

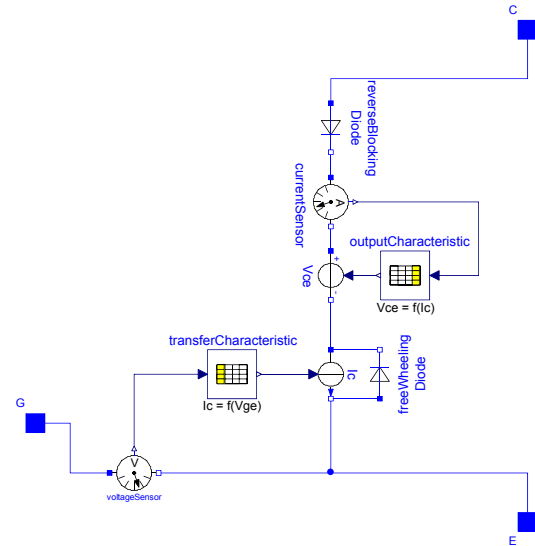


Figure 16: Static IGBT model

$$C_{ge}(V_{ce}) \approx C_{iss}(25V) - C_{rss}(25V) \quad (1)$$

$$C_{gc}(V_{ce}) \approx \frac{C_{rss}(25V) \cdot \sqrt{25V}}{\sqrt{V_{ce}}} \quad (2)$$

Where for instance  $C_{iss}(25V)$  and  $C_{rss}(25V)$  have been determined at a constant collector-emitter voltage of 25V.

Due to the internal semiconductor structure of PT-IGBTs an additional output capacitor  $C_q = f(V_{CE})$  appears which is mainly responsible for the PT-IGBTs switching-off behavior (refer to [4]).

In Figure 17 the dynamic IGBT model is shown. The gate-emitter capacitor is charged over the internal gate resistance  $R1$ . Equation 1 is implemented directly in the parameter window of  $C_{ge}$  as  $C_{iss}$  and  $C_{res}$  are input parameters to the model. Equation 2 is either implemented via equations or a table based model, i.e. the characteristic curve is derived by the equation. Again, temperature dependency has to be introduced to the model (shown for the output characteristic in Figure 17) and the power losses are forwarded as heat flow (not included in the model shown in Figure 17) to the heat port to allow the use of a thermal network.

The main problem of IGBT behavioral models is the lack of information in datasheets regarding tail current. Hence, the tail current has to be measured in advance and fed into the model. Alternatively, if there is no possibility to measure the tail current, the dynamic model can be replaced by the characteristic curves of the switching losses  $E_{on}, E_{off} = f(I_C, T)$ .

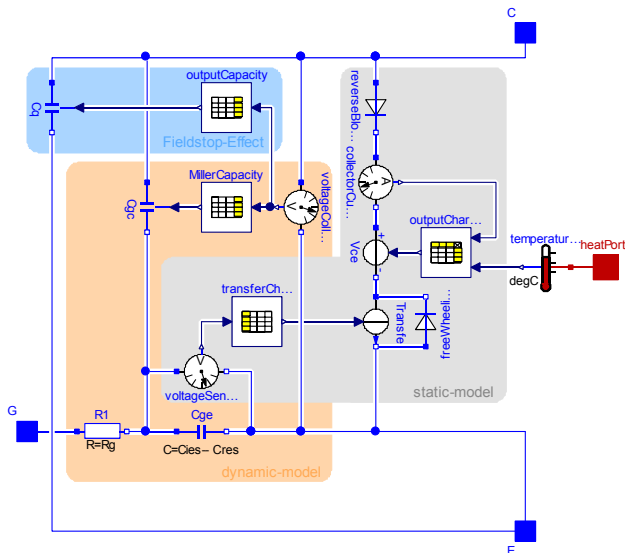


Figure 17: Dynamic PT-IGBT model. If the part highlighted in blue is removed, the model behaves like an NPT-IGBT

## 5 Numerical Performance

The simulation of power semiconductor components faces solvers with a whole series of challenges. To make statements about numerical stability and simulation time, the power-MOSFET model was tested in the three-phase inverter circuit in Figure 12 on its numerical performance<sup>1</sup>.

### 5.1 Integration Algorithm

Detailed modeling of switching operations usually results in so-called *stiff* systems, consisting of differential algebraic equations (DAE) having time constants that differ by several orders of magnitude, e.g. fast switching dynamics and slow thermal processes. These characteristics restrict the selection of the solver enormously. Only solvers based on implicit integration algorithms are suited for such simulations tasks.

Furthermore, switching operations lead to discontinuities in the simulation. Thereby it is distinguished between time- and state-events. In particular the latter causes the solver to iterate for the event, e.g. by using bi-section or regula-falsi algorithms, which increase the simulation time drastically. Even when using an implicit solver, it is not guaranteed that the iterations converge and the simulation succeeds. A deeper insight into this matter is given in [2]. Moreover, if the

<sup>1</sup>any performance comparison was done on an Intel Core i5-3427U CPU, 1.8GHz, 8 GB RAM running Windows 8 (64Bit) and Dymola 2014

switching slopes are not treated ideally, the simulation performance becomes even worse, e.g. real switching slopes are in the region of several 100ns.

### 5.2 Solver Settings

In the upper part of Listing 1, default settings are given which basically match the Dymola standard settings except that the number of intervals is increased due to accuracy reasons. The simulation's performance results, which can be seen in the lower section of the listing, serve as reference values for any following comparison. The impact of the solver settings on the numerical performance is investigated on the basis of the number of function-, hessian- and jacobian-evaluations. At this point it should be mentioned that the parameter for CPU integration time has to be interpreted with caution since it strongly depends on the utilization of the CPU and is therefore not precise in terms of repeatability.

Solver	: DASSL
Number of intervals	: 5.000
Tolerance	: 1e-4
Equidistant time grid	: 0N
Store variables at events	: 0N
-----	
CPU-time for integration	: 253s
Number of GRID points	: 5.001
Number of F-evaluations	: 1.733.187
Number of H-evaluations	: 292.055
Number of J-evaluations	: 145.405
Number of time events	: 9.988
Number of state events	: 5.401

Listing 1: Default solver settings

First, the influence of the number of points plotted into the output diagram is investigated. If the equidistant time grid is deactivated<sup>2</sup>, the solver does not plot the computed values according to the number of intervals parameter, but instead it plots the entire computed values in the output diagram. The number of grid points then increases drastically (228.795) as it was expected whereas the number of F-, H- and J-evaluations basically stay the same. Nevertheless, the number of points that have to be plotted into the output diagram have a big influence on simulation time.

Next, the impact of the solver's integration tolerance is tested. In order to do this, the tolerance is increased by a power of ten. Due to less F-, H- and

<sup>2</sup>simulation setup - output tab - output selection

J-evaluations, the simulation time could be decreased by approximately 25% but again resulting in a loss of accuracy. The same applies to the option 'store variables at events'<sup>3</sup> when deactivated.

In summary, choosing the solver settings in order to optimize the simulation time strongly depends on how accurate the results shall be. The number of intervals should be chosen manually meaning that the equidistant time grid is activated.

### 5.3 Optimization of the Model

For optimizations of the model, first the ideal components are investigated. The model contains four ideal diodes. The on-resistance and off-conductance is set to  $10^{-5}\Omega$  and  $10^{-5}S$  by default. These values were changed to  $10^{-7}$  for the next simulation. The results showed that this can be done without any problems regarding the simulation speed. A few more state events were generated because the system got stiffer. As a matter of fact, the smaller these values are chosen, the more accurate the power loss computation will be.

Also the tables have optimization potential because the interpolation-type influences the simulation performance. The user can choose between linear and continuous interpolation methods, whereas the first method decreases the simulation time slightly and at the same time is holding the number of additional events (due to the non-continuous method) in a negligible range.

Furthermore, the interpolations can be minimized by fitting polynomial functions through the characteristic curves stored in the tables. The challenge hereby is fitting the curves accurately while the order of the polynomial functions is held low. The problem with higher order polynomials is that they drift away drastically outside the fitted range and in terms of decreasing simulation time the opposite effect would occur. An approach hereby could be to split the curve in multiple sections and fit each with a separate polynomial function. When doing this, a polynomial of higher order can be split into multiple polynomials of lower order resulting in an invariant accuracy and a decrease of simulation time.

### 5.4 Generating Table-Based Models for System Simulation

Although the simulation performance can be slightly improved, the models cannot be used in a system sim-

<sup>3</sup>simulation setup - output tab - output selection

ulation, e.g. to answer questions regarding temperature development or maximum driving range. For this purpose table-based models (efficiency maps) have to be provided, i.e. the overall losses at several operating points have to be stored in a table. Such models were already developed and verified for electric machines at Modelon GmbH and are currently developed for inverter models.

## 6 Problems with Behavioral Models

Among the bad numerical performance due to detailed description of the signals several other problems occur.

1. Firstly, datasheets provide values derived under nominal conditions. However, when such models are integrated in an electric powertrain the conditions strongly depend on the topology and are usually not as specified in the datasheet.
2. Secondly, values for parasitics elements are only available in datasheets of modules, e.g. inverter modules developed by the semiconductor manufacturer. The parasitics that occur in a specific layout have to be measured and afterwards integrated in the model (by introducing parasitic resistors, inductors and capacitors) to ensure that the switching behavior is modeled correctly.
3. Moreover, in case of IGBT models neither informations regarding tail current nor the output capacitor that appears in the PT structure are provided.

## 7 Acknowledgments

The models have been developed by our former student - Patrick Denz - during his thesis at Modelon GmbH. Especially the MOSFET model was modified by Patrick such that it covers all modes of operation. Many thanks for the great job at Modelon GmbH. Many thanks to Arendt Wintrich for the great support regarding state-of-the-art IGBT models.

## 8 Conclusion

The presented model of a power-MOSFET covers the static as well as the dynamic behavior of the component. Furthermore, it can be parameterized solely on the basis of characteristic curves and parameters specified in datasheets. The verification in various test circuits delivered reasonable results in static and dynamic

applications. The model was validated with measurement data of a three-phase inverter motor drive system and the maximum relative error between the measured and simulated total power losses lies within 2%.

Since behavioral models of power electronic components produce many events, the simulation becomes slow. It can be accelerated by widening up the integration tolerance, decreasing the number of intervals or not storing variables at events. Also optimizations on the model itself lead to a better simulation performance. However, when integrating the model into a simulation of the entire vehicle, it must be transformed into an efficiency map model which stores the power losses of different operating points in a table.

The IGBT models are way more complex since the tail current has to be measured in advance to model the signals correctly. If it is not possible - for whatever reason - to measure the tail current it is recommended to replace the dynamic model with tables storing the particular switching losses. This in turn increases the simulation performance and ensures that the losses are modeled correctly.

## References

- [1] Infineon Technologies AG. *Automotive IGBT Module - Explanation of Technical Information*. 2010.
- [2] Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.
- [3] Alan Courtay. *MAST Power Diode and Thyristor Models Including Automatic Parameter Extraction*. CERN, European Laboratory for Particle Physics. Geneva, Switzerland, 1995.
- [4] Bayerer Reinhold Heer, Daniel and Thomas Schütze. *Trench Field-Stop IGBT3 Turn-Off*. Infineon Technologies AG, Warstein, Germany, 2012.
- [5] Josef Lutz. *Halbleiter-Leistungselemente: Physik, Eigenschaften, Zuverlässigkeit*. Springer, 1st edition, 2006.
- [6] Martin Otter, Hilding Elmqvist, and Sven Erik Mattsson. *Modeling of mixed Continuous/Discrete Systems in Modelica*. 1999.
- [7] Frank Pfirsch and Reinhold Bayerer. *MOS-gesteuerte Leistungsschalter: Konzepte und Schaltverhalten*. VDE-Verlag, Bauelemente der Leistungselektronik und ihre Anwendungen, ETG-Fachtagung, 2006.
- [8] Arendt Wintrich. *Verhaltensmodellierung von Leistungshalbleitern für den rechnergestützten Entwurf leistungselektronischer Schaltungen*. PhD thesis, Technische Universität Chemnitz, 1996.
- [9] Arendt Wintrich, Ulrich Nicolai, Werner Tursky, and Tobias Reimann. *Applikationshandbuch Leistungshalbleiter*. ISLE, Ilmenau, 2010.



# Verification and Design Exploration through Meta Tool Integration with OpenModelica

Zsolt Lattmann<sup>2</sup>, Adrian Pop<sup>1</sup>, Johan de Kleer<sup>3</sup>, Peter Fritzson<sup>1</sup>, Bill Janssen<sup>3</sup>,  
Sandeep Neema<sup>2</sup>, Ted Bapty<sup>2</sup>, Xenofon Koutsoukos<sup>2</sup>,  
Matthew Klenk<sup>3</sup>, Daniel Bobrow<sup>3</sup>, Bhaskar Saha<sup>3</sup>, Tolga Kurtoglu<sup>3</sup>

<sup>1</sup>Department of Computer and Information Science

Linköping University, SE-581 83 Linköping, Sweden

<sup>2</sup>Vanderbilt University, Nashville, TN, USA

<sup>3</sup>Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 USA

lattmann@isis.vanderbilt.edu, {adrian.pop, peter.fritzson}@liu.se, dekleer@parc.com

## Abstract

Modelica models are typically used for simulation to investigate properties of a possible system designs. This is often done manually or combined with optimization to select the best design parameters.

It is desirable to have systematic and partly automated support for exploration of the design space of possible designs and verifying their properties vs. requirements. The META design tool chain is being developed to support this goal. It provides an integration framework for components, designs, design spaces, requirements, and test benches, as well as verification of requirements for the generated design models during design exploration

This paper gives an overview of the META tools and their integration with OpenModelica. The integrated environment currently has four main uses of OpenModelica: importing Modelica models into the META tool model structure, performing simulations within test benches, analyzing Modelica models and automatically adding fault modes, and extracting equations (DAEs) for formal verification tools, e.g. the QRM using qualitative reasoning.

A prototype of the integrated tool framework is in operation, being able to generate and simulate thousands of designs in an automated manner.

*Keywords: Modelica, simulation, design exploration, verification, etc.*

## 1 Introduction

A design tool chain (META tools, Figure 1) is being developed for exploring design alternatives under cer-

tain condition and to verify their properties versus formalized requirements.

A design is built from component model building blocks defining component dynamic behavior and is defined as a composition of component models. A design space can represent different component alternatives as well as different design architectures.

After a design or design space has been created, test cases can be defined against the given requirement set. The test cases, which are called *test benches*, are executable versions of the system requirements.

From the test bench models, the META tools can compose analysis packages over a design space for different domains such as simulation of DAEs (differential algebraic equations), formal verification, static analysis, and structural analysis.

The integrated environment currently has four main uses of OpenModelica: importing Modelica models into the META tool model structure, performing simulations within test benches, analyzing Modelica models and automatically adding fault modes, and extracting equations (DAEs) needed for formal verification tools.

## 2 The OpenMETA Tool Chain

The OpenMETA<sup>1</sup> tool chain is being developed under DARPA's Adaptive Vehicle Make (AVM) [5] program that contains a set of projects one of them is the META project. The AVM program aims to reduce vehicle design and manufacturing time using the framework and toolset provided by the META program.

---

<sup>1</sup> Provided under MIT license

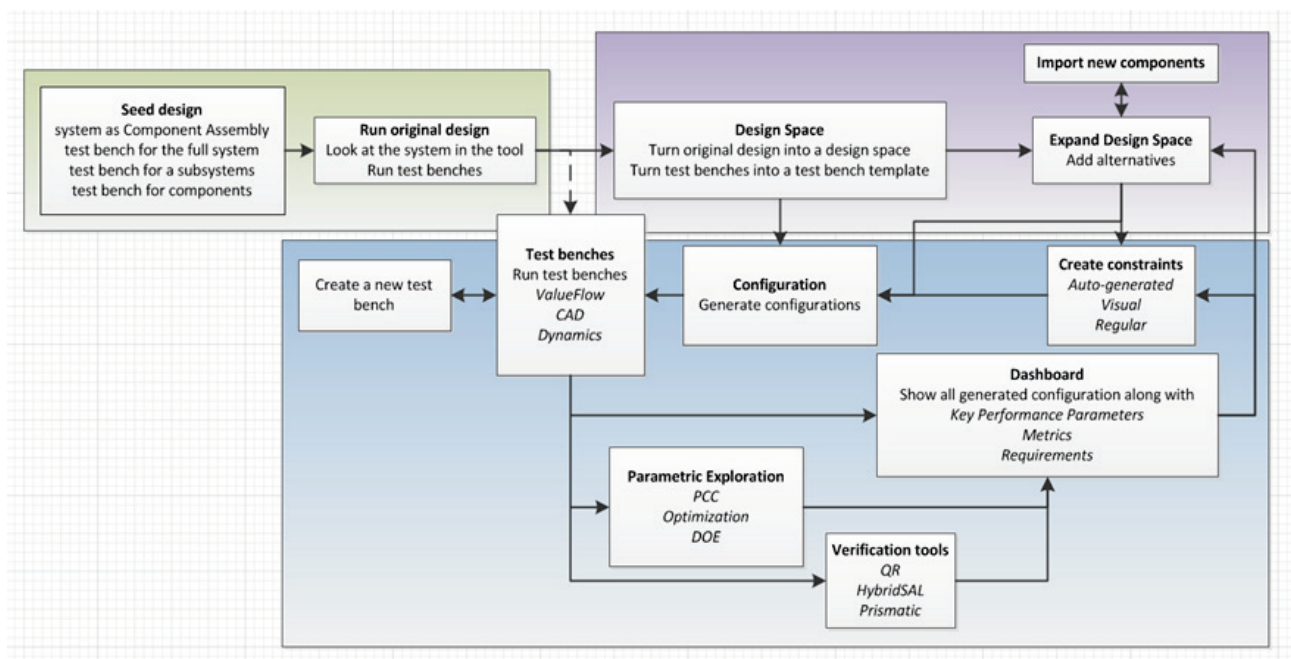


Figure 1. Design flow in the OpenMETA Tool Chain.

The tool chain consists of a language/meta-model called Cyber Physical Modeling Language (*CyPhyML*), a set of *model transformation software* components that translate from *CyPhyML* models to various domain tools, an analysis package executor (referred to as *Job Manager*), and a visualizer (referred to as *Project Analyzer*) for inspecting and understanding the results of analysis packages.

We present (a) the concepts defined in *CyPhyML* in Section 2.1, (b) the integration points with and utilization of OpenModelica in Section 2.2 and Section 2.3, (c) collected analysis results in Section 2.4 and Section 2.5, and (d) the usage of formal verification methods in Section 2.6.

## 2.1 Concepts

*CyPhyML* is a Domain Specific Modeling Language (DSML) built for modeling cyber, physical, and manufacturing component models, composing the component models, making architecture trade-offs using design spaces, and encoding test cases for various analysis domain tools. *CyPhyML* is defined using the MetaGME language in the Generic Modeling Environment (GME [6]).

A *CyPhyML* Component model contains interfaces (physical, structural, and data) of a physical entity or a controller, key parameters of the component, and the relationship between component level parameters and domain model parameters. For instance, a mass component can have a *manufacturing domain model*, a *geometric domain model* (CAD), and a *behavior domain model* (Modelica model).

The component model level parameters can affect all domain model parameters at the same time, i.e., if the mass has dimension and density parameters, then the CAD model and the behavior model are parameterized with exactly the same values respecting unit conversions.

When the CAD and behavior models are composed, all parameters will be consistent across all domain models. *CyPhy* Component models do not contain any internal details of the domain models; they capture information only about interfaces and links to the domain models.

A *CyPhy Component Assembly model* can contain any number of *CyPhyML* Components and other *CyPhyML* Component Assemblies, which together provide system and subsystem concepts. This language feature makes hierarchical composition possible through interfaces (ports and parameters). A full system model is often called a point design or a single design configuration.

A *CyPhyML Design Space model* can encode multiple design configurations (i.e., component assemblies) by using alternative and optional containers inside the design space. Design space models generate a discrete design space in the form of design configurations using the Design Space Exploration Tool (DESERT [5]).

For instance, if the design space contains a mass component, alternative mass components can be added (e.g. using different geometric sizes, material, etc.); if 3 options are added for the mass component, the design space will grow to 3 design configurations. If we have a mass, spring, and damper system (similar to a very simplified suspension assembly) and 3 options are

available for each, then the overall design space would be 27 configurations.

To solve the design space exploration problem, CyPhyML supports design space constraints that can be expressed as auto-generated range constraints, property constraints (e.g. component level parameter limits), visual constraints (e.g. compatibility between components/material or symmetry), or as Object Constraint Language (OCL) constraints. Constraints are used to prune the exponentially large combinatorial design space to a feasible and manageable set of configurations.

Once a CyPhyML Design or Design Space is built, we can define the evaluation of designs using CyPhyML Test Bench models. CyPhyML Test Benches are used to set up boundary and environmental conditions for designs in which they should be evaluated. Test benches also provide sufficient information and any additional models (e.g. stimulus, load, external ‘test’ components) to the system to make simulation and analysis possible with a domain-specific tool.

CyPhyML supports various types of test benches, including Dynamics (i.e., Modelica simulations), formal verification, CAD (e.g. composing the 3D model and computing center of gravity or mass), finite element analysis, computational fluid dynamics, blast, ballistic, conceptual manufacturing, detailed manufacturing, and reliability analysis.

In this paper we focus on *formal verification* and Dynamics (Modelica) *simulation test benches* only. CyPhyML Test benches contain a top level system under test (design or design space), input parameters that can change environment, load, stimulus conditions (test component parameters), and outputs called metrics.

## 2.2 Importing Modelica Models

CyPhyML Components have associated behavior models in the form of linked Modelica models. Only Modelica parameters and Modelica connectors need to be represented in the CyPhyML Component model. The behavioral model aspect of a CyPhyML Component can be viewed as a lightweight wrapper around a Modelica model, which can be built using the OpenMETA tool set and its editor GME.

Building the Modelica model interface representation in GME can be cumbersome and a time consuming activity. All information about the interface exists in the Modelica model, already including the following: model name, model type, connector names, connector types, parameter values (e.g. default value, minimum value, and maximum value), and class restrictions.

The user has to provide a set of Modelica models in textual form (.mo files or one .mo package). A wide set

of Modelica models can be imported in an automated way as CyPhyML Components or CyPhyML Test Components using the OpenModelica Compiler (OMC) API. There is a seamless integration between the OpenMETA tools and the OMC API. The OMC API provides functionality to load model files and libraries (i.e., packages), query containment and inheritance relationship between types, and navigate through model elements using the abstract syntax tree.

The Modelica model importer has certain limitations and it does not support the entire Modelica language. Conditional ports and parameters, enumerated types, and parameterized ports (which can change their internal structure) are not supported. ‘Replaceable’ elements have a limited support, for instance models with fluid port connectors can be imported and the ‘Medium’ type is correctly set in the CyPhyML Component.

If the model or library does not conform to the Modelica Specification and/or the OMC API cannot load the package, then the automated import functionality is not available in the OpenMETA tools, requiring users to build the CyPhyML Components manually.

We are currently working on supporting a more complete set of the Modelica language and multi-fidelity models where one CyPhyML Component can be linked to more than one Modelica model and where the different Modelica models represent different level of modeling abstraction of the behavior of the physical component.

The OpenMETA tools already have a limited support for multi-fidelity component models, but they have to be built manually. For any CyPhyML Test Bench the component fidelity selection can be specified for a class of components, e.g., spring or damper component models.

## 2.3 Generating Modelica Models

Once a set of Modelica components are imported into the CyPhyML we can build design models, design space models and test-bench models. These models are composed through interfaces (i.e., connectors and parameters), which is sufficient information to generate composed Modelica models of test bench models for a design or for the entire design space.

The generated Modelica models preserve the hierarchical decomposition of the system and organize all generated models into packages and sub-packages based on the CyPhyML project structure to ease navigation in the generated model. Each generated component model, used in the design, ‘extend’ the referenced Modelica model and overwrites the parameters with the CyPhyML Component instance values.

From each dynamics CyPhyML Test Bench there are two generated models in the Modelica package that are used to run analysis. One of them is a simulation model and the other one is an augmented version of the simulation model for formal verification purposes.

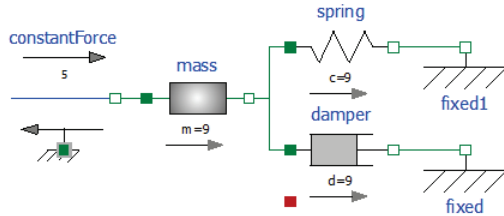


Figure 2. Mass-Spring-Damper in Modelica

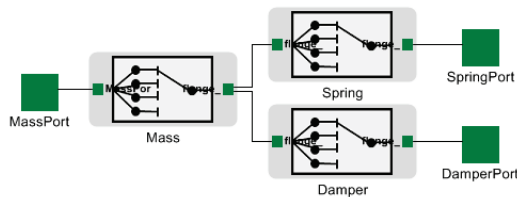


Figure 3. Mass-Spring-Damper design space in OpenMETA tools

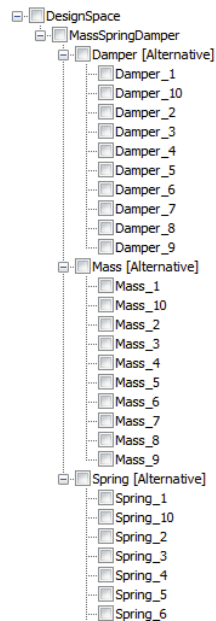


Figure 4. Mass-Spring-Damper design space tree and alternative options

We use a Mass-Spring-Damper (MSD) system, which contains Modelica Standard Library components, as a simple use case to show the workflow and the results of the formal verification tool for two configurations from a CyPhyML Design Space. Modelica model of the MSD system structure is shown in Figure 2. Figure 3 depicts the design space in the OpenMETA tools, where each component *Mass*, *Spring*, and *Damper* contains alternative components. The hierarchical structure and alternative options are shown in Figure 4. There are

10 alternatives in each design container, thus the design space generates  $10 \times 10 \times 10$  (1000) configurations. We have selected two configurations (#1 and #8) for further analysis by the verification tool.

Configuration #1 and configuration #8 have the same architecture, i.e., the same number and kind of components and the same connections among the components, but configuration #1 uses Mass 9 ( $m=9$  kg), Spring 9 ( $c=9$  N/m), Damper 9 ( $d=9$  N.s/m) and configuration #8 uses Mass 3 ( $m=3$  kg), Spring 8 ( $c=8$  N/m), Damper 8 ( $d=8$  N.s/m).

Figure 5 shows configuration #1 of a generated Modelica model for formal verification. The verification model inherits the simulation model and includes: the definition of the requirement status (success, unknown, violated), all physical limit definitions (e.g. `Limit1`: maximum absolute force cannot exceed a certain value) and requirements for the system, and defines all conditions under which the limits (e.g. `abs(Spring.f) > 17`) and requirements are violated.

```

model verif_MassSpringDamperAuto2cfigs_cfg1
  extends CyPhy.TestBenches.MassSpringDamperAuto2cfigs_cfg1;

  // Requirement Definition
  type Requirement = enumeration(
    success,
    unknown,
    violated);

  // Limit-Checks definitions
  Requirement Limit1(start = Requirement.unknown, fixed=true)
  " Max absolute value limit-check on
  |MassSpringDamper.Spring_1__Spring_9.flange_a.f";

  // Requirements-Checks

equation
  // Limit-Checks equations
  if Limit1 == Requirement.violated or
  |MassSpringDamper.Spring_1__Spring_9.flange_a.f > 17 or
  |MassSpringDamper.Spring_1__Spring_9.flange_a.f < -17 then
    Limit1 = Requirement.violated;
  else
    Limit1 = Requirement.unknown;
  end if;

  // Requirement-Metrics Checks equations

end verif_MassSpringDamperAuto2cfigs_cfg1;
    
```

Figure 5. Modelica model of MSD configuration #1.

## 2.4 Model translators and Job Manager

CyPhyML analysis model translators (i.e., analysis interpreters) are built to generate analysis packages from CyPhyML test benches, which contain domain tool specific input files, data structures, and scripts to perform the execution and collect results.

The OpenMETA tools can generate analysis packages for all test benches over the entire design space. This raises another scalability issue: executing all analysis packages may take significant time. In order to reduce the overall runtime, the META Job Manager [8] can run the individual/independent analysis packages in parallel either locally utilizing multiple CPU cores, or

on a remote compute cloud provided by the VehicleFORGE [8] platform. After analyses are executed, the results are stored locally.

### 2.5 Analysis Results

The raw analysis results are cumbersome and can be extremely difficult to compare. To address this issue, CyPhyML defines metrics for the key performance parameters of design configurations. These numbers, which are often driven by system requirements, provide the basis for design trade-offs and ranking, as well as for making decisions under specific circumstances about which design configuration is best.

Metrics are stored in a manifest file that contains the key information about the design configuration, test bench, and the projection of results. This single file is much smaller than the raw data and makes design space comparisons significantly easier. In general, we noticed roughly a three orders of magnitude size reduction when using only the compact manifest file (e.g. 15 GB of raw analysis data -> 10 MB).

The OpenMETA tools provide a data visualizer called the Project Analyzer. The Project Analyzer can be used locally in a web browser or deployed on VehicleFORGE (or another server). It loads all analysis data from the manifest files (no data from raw files is loaded) and provides different visualization techniques to display results, visualize requirements, rank designs based on the user's weighting preference on metrics, show physical limit violations on components, display constraint plots, show formal verification results, etc.

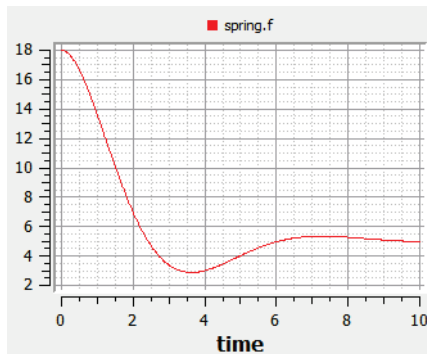


Figure 6. Mass-Spring-Damper simulation results configuration #1 force on the spring component

OpenModelica can be used to visualize the raw simulation results if needed. Figure 6 shows the force [N] on the spring component for design configuration #1.

The Project Analyzer provides various visualization techniques using different widgets to visualize the results over a design space. Figure 7 shows the parallel axes plot widget, where the vertical axes correspond to the metrics (velocities) and each colored line between the axes represent a design configuration. The require-

ments objective and threshold values are shown on the right hand side of the axes.

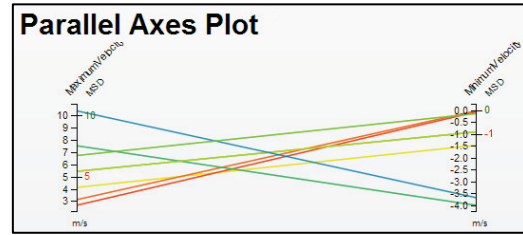


Figure 7. Project Analyzer parallel axes plot

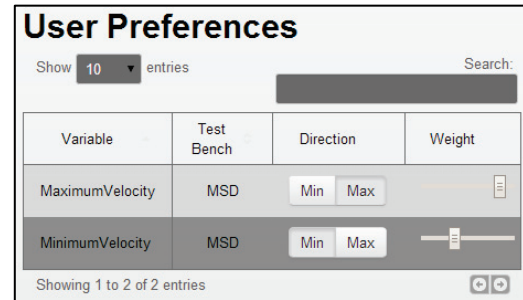


Figure 8. Project Analyzer user preferences settings

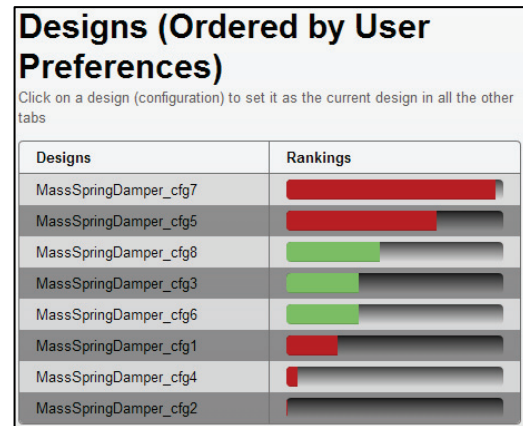


Figure 9. Project Analyzer designs by user preferences and color coded based on requirements

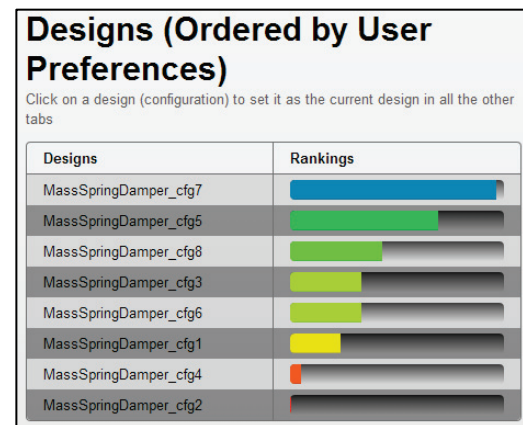


Figure 10. Project Analyzer designs by user preferences and color coded based on ranking

Users can set their weighting preference (Figure 8) for each metric value, which would determine the ranking

of the designs shown in Figure 9 and Figure 10. Designs on each widget can be color coded based on requirements (Figure 9), ranking (Figure 10), limit violations, or design scores.

## 2.6 Simulation and Verification of Generated Modelica Models

We have chosen to separate the verification and the simulation models in the generated code. We aim to run the simulations as fast as it possible, since running the test benches over a design space can take significant time even if parallel execution is used.

Using the OpenMETA tools and the parallel execution capability provided by the Job Manager we run hundreds of design configurations over tens of test benches. The simulation model does not need to contain verification properties and unnecessary auxiliary variables. Therefore, the simulation can first be executed, and then a post processing script can validate the limit violations and requirements on the simulation results.

This approach will give us a faster execution time for simulation models. The OpenMETA tools use OpenModelica to execute the generated simulation models.

Modelica models for verification are translated to Differential Algebraic Equations using the OpenModelica Compiler. The Mass-Spring-Damper configurations are translated to DAEs and then a formal verification tool analyzes both configurations.

```

"AnalysisStatus": "OK",
"TierLevel": 0,
"DesignName": "MassSpringDamperAuto2cfigs_cfg1",
"FormalVerification": [
  {
    "Source": "PARC",
    "Result": "FAIL",
    "ReqName": "Limit1",
    "Details": [
      {
        "GroupBody": [
          "Decrease MassSpringDamper.Spring_1__Spring_9.c",
          "Decrease roof.s0",
          "Increase floor.s0",
          "Decrease MassSpringDamper.Spring_1__Spring_9.s_rel0",
          "Increase MassSpringDamper.Damper_1__Damper_9.s_rel"
        ],
        "GroupTitle": "Changes suggested by QRM's symbolic differe"
      }
    ]
  }
],

```

Figure 11. Verification results for MSD configuration #1.

```

"AnalysisStatus": "OK",
"TierLevel": 0,
"DesignName": "MassSpringDamperAuto2cfigs_cfg8",
"FormalVerification": [
  {
    "Source": "PARC",
    "Result": "UNKNOWN",
    "ReqName": "Limit1",
    "Details": [
    ]
  }
],

```

Figure 12. Verification results for MSD configuration #8.

The limit restrictions and requirements are the same for configuration #1 and #8. Figure 11 and Figure 12 depict the results of the formal verification results for configuration #1 and configuration #8 respectively. Section 3 and Section 4 describes the integrated formal verification method and reliability analysis in more detail respectively.

## 3 Qualitative Reasoning Module

The Meta tool suite includes a Qualitative Reasoning Module (QRM) which performs qualitative analyses of system behavior. In contrast to Modelica solvers which produce exact numerical results given exact numerical inputs and parameter values, qualitative simulators predict the possible time evolution of a system in qualitative terms.

Qualitative values are characterized by ranges, for example  $Q^+$  represents any possible positive value. Qualitative values can be demarcated with landmarks, for example  $[l, u]$  where the value lies between  $l$  and  $u$ . A qualitative analysis may show that a particular design cannot ever meet its requirements—something that is impossible to show with numerical solvers.

One challenge to more widespread use of Qualitative Reasoning is the lack of extensive qualitative model libraries. One cannot expect a designer to write their own qualitative models. Therefore we have spent considerable time and effort into automatically translating Modelica models into terms suitable for qualitative analysis.

Our translator starts with the exported DAE from OpenModelica. This has required significant extensions of model importers to qualitative algorithms. In addition, the DAE exporters have had to be extended to provide additional information. Qualitative reasoning requires declarative models. Any Modelica model used by AVM which is not purely declarative is being converted to declarative form manually.

Qualitative reasoning is most useful in early stages of conceptual design where the parameters, topologies, and requirements have not been completely articulated. Topologies which cannot possibly achieve customer requirements can be eliminated without having to determine specific parameters.

Qualitative analysis can also analyze a design which fails to meet some requirement and suggest qualitative parameter changes which will bring the design closer to meeting a requirement. The screenshot (Figure 13) illustrates analyzing a mass-spring-damper system to identify qualitative changes to meet a requirement.

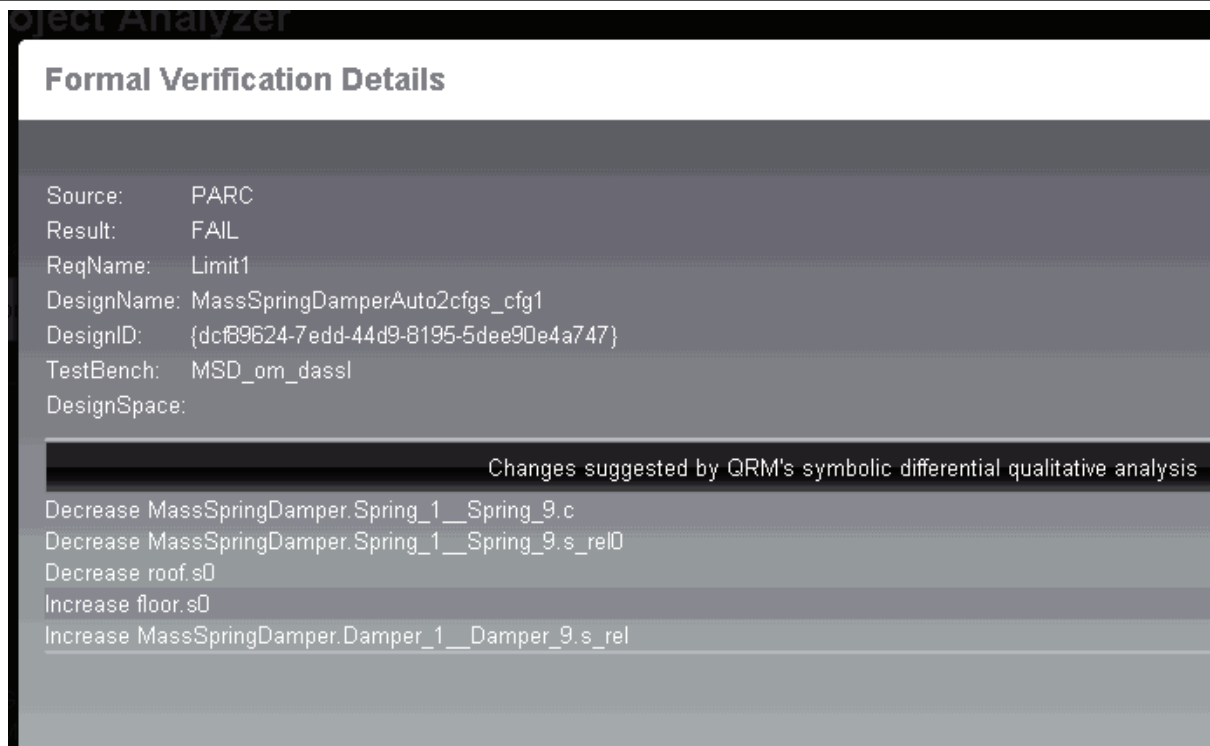


Figure 13. Using QRM for analysis of a mass-spring-damper system.

## 4 Reliability Analysis

The Meta tool suite includes a reliability analysis tool. This tool automatically allows a designer to evaluate the reliability of various components as well as various design configurations. The reliability tool has three major modules: (1) automatic construction of Modelica fault models, (2) determination of the fault probability distributions, (3) computing system reliability given (1) and (2).

Our fault augments takes a MSL model as input and automatically constructs its fault modes, which includes power port failures such as open and shorts as well as important parameter shifts. For each fault model, we construct damage maps which provide a probability density function for important parameters and is indexed by the type of material the particular component is constructed out of (e.g., steel), CAD properties, and Modelica variable values. The damage maps are constructed through a separate probabilistic process. More details can be found in [13]. In this paper we will focus on how the reliability tool is used by a designer (i.e., the third module).

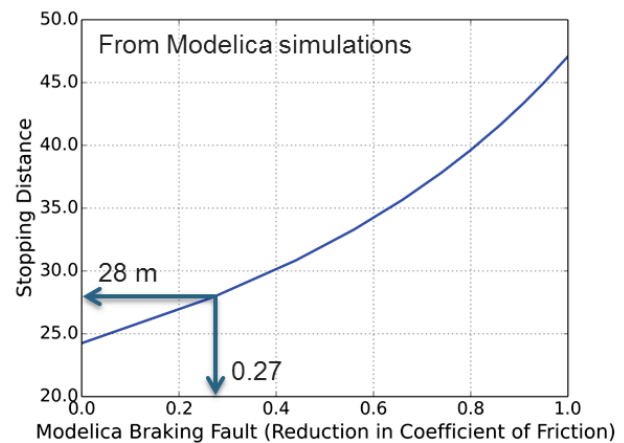


Figure 14. Braking distance / the coefficient of friction

Suppose a designer needs to choose brake in their design (vehicle drive train) that will meet its stopping requirement of 28 m from 60 kph. Given a fault augmented model, we can determine stopping distance by running multiple Modelica simulations.

From the Modelica simulations we can see that the stopping criterion fails after a fault amount of 0.27. Using the reliability formula and given a reduction in friction the actual physical damage is 0.85.

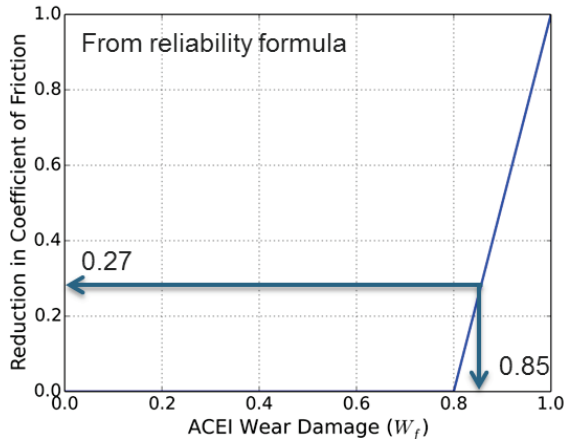


Figure 15. ACEI Wear damage / coeff. of friction.

Finally we refer to the damage map to determine the probability that the vehicle will meet its braking requirement after 75 missions.

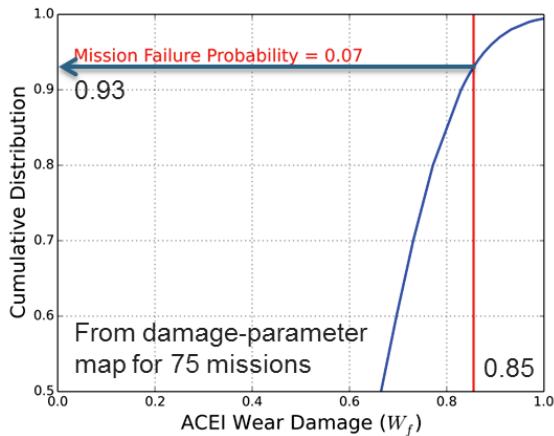


Figure 16. ACEI Wear damage / cumulative distribution.

With the reliability tool the designer can choose the component, requirement, number missions, desired probability of success, etc.

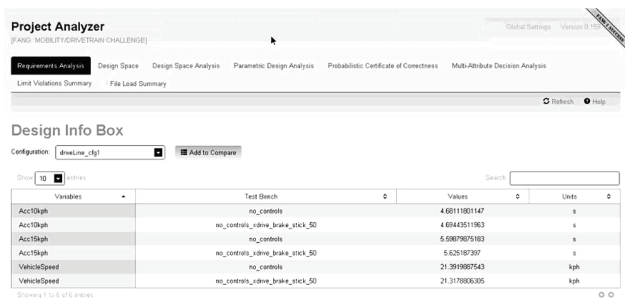


Figure 17. Reliability tool.

The needed Modelica simulations required to render these reliability calculations are expensive. Our approach is to pre-compute as much as possible. For example, the damage maps are all pre-computed. For the simple vehicle model analysis presented here we have

pre-computed all Modelica simulations (and use interpolation) to enable the reliability tool to respond instantly. However, for complex novel designs the reliability calculations will take hours and possibly days on a single machine. Fortunately, reliability calculations scale linearly with the number of processors.

## 5 OpenModelica Tool Support

OpenModelica is used in four different places in the OpenMETA tool chain:

- importing Modelica models and associating them with CyPhyML component models,
- performing simulations of composed Modelica models, i.e., CyPhyML test benches,
- analyzing Modelica models and automatically adding fault modes, and
- extracting Differential Algebraic Equations (DAEs) for formal verification tools.

The OpenModelica compiler (Figure 18) has been slightly extended to facilitate integration with the META Tool chain.

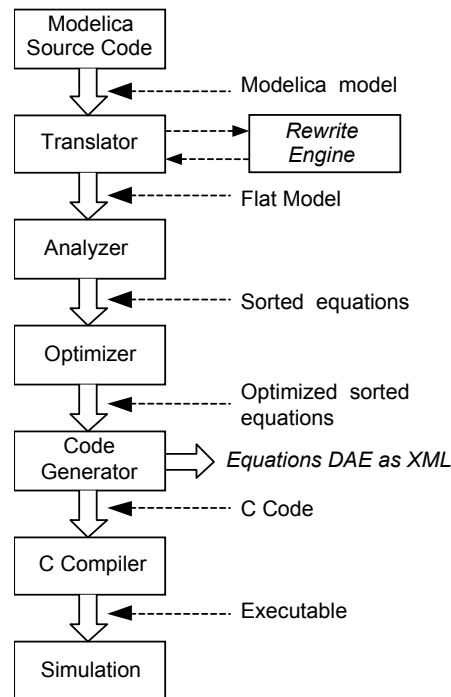


Figure 18. The OpenModelica compiler (OMC) structure and simulation execution. A rewrite engine and enhanced DAE XML output have been added for Meta Tool usage.

### 5.1 User-defined Rewrite Rules for Model Simplification

In order to make it feasible to apply formal verification to models they need to be simplified as much as possible so that their complexity is drastically reduced.



To support model simplification a *rewrite engine* for user-defined rewrite rules has been added to the OpenModelica compiler (Figure 18).

Note that this simplification could be applied by the formal verifier tool on the final DAE, but if the rewrite rules are applied as early as possible inside the Modelica compiler further simplifications can be discovered and applied.

The final model representation form as reduced and optimized symbolic equations is output in an XML representation for further processing by the Meta Tools QRM module.

The user defined rewrite rules have the form:

```
rewrite(old_expression, new_expression);
```

Note that `old_expression` and `new_expression` can contain special component references in the form of quoted identifiers starting with \$, for example: '\$1', '\$2', '\$x', '\$y', etc.

The part of the expression tree where the special component reference appear is bound to that component reference.

As an example, consider the rule:

```
rewrite(
  abs('$1'),
  if ('$1' >= 0) then '$1' else -'$1');
```

which could be applied to an expression:

```
abs(y + z)
```

In this case \$1 will be bound to  $y+z$  and the transformed expression becomes:

```
if ((x+y) > 0) then (x+y) else -(x+y)
```

The bounding operation is similar to pattern matching or unification in languages that support such features. Some examples of user-defined rewrite rules:

```
rewrite(
  abs('$1'),
  if ('$1' >= 0) then '$1' else -'$1');

rewrite('$1' ^ 2, '$1' * '$1');

rewrite(semiLinear(0.0, '$1', '$2'), 0.0);

rewrite(noEvent('$1'), '$1');

rewrite(
  Modelica.Fluid.Utilities.regStep(
    '$1', '$2', '$3', '$4'),
  if ('$1' > '$4') then '$2' else '$3');
```

The rules are loaded from a file given by the user and the rules are matched/applied to the expressions appearing in the abstract syntax tree.

Note that the application of the rules happen during semantic checking of expressions so that the resulting type before and after the application of the rule can be

checked. In the cases where the bound expressions are arrays the operation is applied for each element, for example:

```
rewrite(
  Modelica.Math.Matrices.isEqual('$1',
    '$2', '$3'),
  '$1'=='$2');
```

applied to:

```
Modelica.Math.Matrices.isEqual({{x,y},
  {z,w}}, {{a,b},{c,d}}, eps)
```

will result in:

```
x == a and y == b and z == c and w == d
```

One can see that in some cases not all variables are used as for example `eps` above. For the purpose of formal verification the given expression is enough as the `eps` is used only for robustness of simulation.

## 6 Integrated OpenModelica Meta Tools Environment

The following summarizes the main capabilities of the integrated OpenModelica – META Tools environment:

- Parse Modelica models (OpenModelica compiler API called through Python) and import model interfaces (parameters and connectors) into the META tool chain.
- Run simulations of composed Modelica models using the OpenModelica (OMC compiler).
- Be able to formally *evaluate verification properties* of system designs using OpenModelica and verification tools QR/HybridSal (XML DAE).
- OpenMETA tools can *compose simulation models over a design space* including different architecture variations in an automated way.
- Verification problems and simulation models can be encoded as *test bench*, which can be evaluated over a *design space*.
- Using the OpenMETA tools the *JobManager* provides sufficient capabilities to utilize all CPU cores in the user's computer.
- The analysis simulation/verification can run locally or on a remote execution cluster.
- Simulation and verification results are collected and visualized through a common interface called *Project Analyzer*.

## 7 Related Work

Automated verification of dynamic behavior of design models against formalized requirements is described in

[10] and [11]. A prototype of an integrated tool chain for model based functional safety analysis is presented in [12].

## 8 Conclusions

This paper has presented an overview of the META tools for design space exploration and design verification, and their integration with OpenModelica.

The integrated environment currently has four main uses of OpenModelica: importing Modelica models into the META tool model structure, performing simulations within test benches, analyzing Modelica models and automatically adding fault modes and extracting equations (DAEs) for formal verification tools, e.g. the QRM using qualitative reasoning.

A prototype of the integrated tool framework is in operation, being able to generate and simulate thousands of designs in an automated manner.

## 9 Acknowledgements

This work was partially sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program and is Approved for Public Release, Distribution Unlimited. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

This work has also been partially supported by the Swedish Governmental Agency for Innovation Systems (Vinnova) within the ITEA2 MODRIO project, and by the Swedish Research Council (VR).

## References

- [1] Modelica Association. *Modelica—A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.2 rev 2*. Available at <http://www.modelica.org>, August, 2013.
- [2] Modelica Association. *Modelica Standard Library 3.2 rev 1*. <http://www.modelica.org>. Aug. 2013.
- [3] Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*, ISBN 0-471-471631, Wiley-IEEE Press. 2004.
- [4] Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3*, Accepted for Publication, Wiley-IEEE Press. 2004.
- [5] Adaptive Vehicle Make. [http://www.darpa.mil/Our\\_Work/TTO/Programs/Adaptive\\_Vehicle\\_Make\\_\(AVM\).aspx](http://www.darpa.mil/Our_Work/TTO/Programs/Adaptive_Vehicle_Make_(AVM).aspx)
- [6] A. Ledeczki, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomasson, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment", Workshop on Intelligent Signal Processing, Budapest, Hungary, May, 2001.
- [7] Sandeep Neema, J. Sztipanovits, G Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis". In *Embedded Software*, R. Alur and I. Lee, eds., pp. 290–305, Vol. 2855 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003.
- [8] Laszlo Juracz, Zsolt Lattmann, Tihamer Levendovszky, Graham Hemingway, Will Gaggioli, Tanner Netterville, Gabor Pap, Kevin Smyth, Larry Howard. VehicleFORGE: A Cloud-Based Infrastructure for Collaborative Model-Based Design., In *Proc. of 2nd International Workshop on Model-Driven Engineering for High Performance and CCloud computing (MDHPCL)*, MODELS 2013, Miami, FL, USA, 2013.
- [9] Raj Minhas, Johan de Kleer, Ion Matei, Bhaskar Saha, Daniel G. Bobrow and Tolga Kurtoglu. Using Fault Augmented Modelica Models for Fault Diagnostics. Submitted to Modelica'2014. Dec 2013.
- [10] Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements - Method, Language, and Tool*. Linköping Studies in Science and Technology, Dissertation No. 1547, [www.ep.liu.se](http://www.ep.liu.se), Nov 12, 2013.
- [11] Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan Paredis. Virtual Verification of System Designs against System Requirements. In *Proc. of 3rd International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES'2010)*. In conjunction with MODELS'2010. Oslo, Norway, Oct 4, 2010.
- [12] Lena Rogovchenko-Buffoni, Andrea Tundis, Muhammed Zoheb Hossain, Mattias Nyberg, Peter Fritzson. An Integrated Tool chain For Model Based Functional Safety Analysis. Accepted to *Journal of Computational Science*, June, 2013.
- [13] Johan de Kleer, Bill Janssen, Daniel G. Bobrow, Tolga Kurtoglu, Bhaskar Saha, Nicholas R. Moore and Saravan Sutharshana, Fault Augmented Modelica Models, *24th International Workshop on Principles of Diagnosis*, Jerusalem, pp. 71-78, 2013.

# Parallel Model Execution on Many Cores

Hilding Elmqvist    Sven Erik Mattsson    Hans Olsson  
Dassault Systèmes

Ideon Science Park, SE-223 70 Lund, Sweden

Hilding.Elmqvist@3DS.com    SvenErik.Mattsson@3DS.com    Hans.Olsson@3DS.com

## Abstract

Modelica gives the possibility to compose more and more detailed models since model components can be reused. This means that simulation needs to be faster. One possibility is then to use multi-core technology. Recent advances with more than 1000 cores show the potential.

The problem is then how to utilize this enormous processing power in a user friendly way. Partitioning needs to be made automatically. Modelica gives good possibility to automatically partition the model equation execution into separate threads since it is a declarative language based on equations.

This paper describes a method to automatically parallelize model equations implemented in Dymola. A speed-up of 3.4 times has been achieved using 4 cores/8 threads.

*Keywords: Modelica; Multi-core; Automatic partitioning*

## 1 Introduction

Modelica gives the possibility to compose more and more detailed models since model components can be reused. This means that simulation needs to be faster. One possibility is then to use multi-core technology. Recent advances with more than 1000 cores show the potential. One example is the Kalray MPPA (Multi-Purpose Processor Array) Technology (<http://www.kalray.eu/>). The MPPA 256 has 256 cores on one chip.

The problem is then how to utilize this enormous processing power in a user friendly way. Partitioning needs to be made automatically. Modelica gives good possibility to automatically partition the model equation execution into separate threads since it is a declarative language based on equations. (Aronsson, 2002, 2006) discussed automatic partitioning algorithms but speed-up was only achieved in special cases.

This paper describes technology to automatically parallelize model equations. Results using the implementation made in Dymola are included. Thermodynamic and electrical examples are discussed in detail. A speed-up of 3.4 times has been achieved using a laptop with Intel(R) Core i7-3740QM CPU @ 2.70GHz with 4 cores/8 threads.

## 2 Algorithms for Parallelization

The goal is to make a static scheduling of the equation execution. To find the optimal schedule is very complex, so various heuristics needs to be utilized. The first phase is to investigate which equations that could be executed in parallel. In this phase, the number of possible threads is not utilized.

In order to balance the efforts of executing the parallel sections, we need to have some estimates of the cost of executing different parts. Thus, the equations are analyzed and manipulated in the ordinary way until we have a sequence of calculating derivatives and variables at a given point of time. The calculations may include simple assignments, calls to Modelica functions or external functions and solving linear, nonlinear, mixed discrete/real systems of equations. An element in this sequence will be called a block node and it includes a set of solved equations and a set of variables that are solved by the node. The ordering of the block nodes is typically obtained by making a Block Lower Triangular (BLT) partition of the problem. The BLT partitioning defines a partial ordering meaning that a node can be executed if all block nodes with lower numbers have been executed. There may be other orderings that also may be feasible. If two successive blocks do not need each other's result, the order can be exchanged. They can in fact be executed in parallel. The major steps to investigate what can be executed in parallel are:

1. Build a dependency graph including all block nodes. Let the edges represent the dependencies between the nodes such that if node  $N_i$  needs the input of a variable  $v$  be-

- longing to node  $N_j$  there is an edge between  $N_i$  and  $N_j$ . We can make the edges directed:
- a. For  $N_i$  it is a “need” edge meaning that it needs the results of  $N_j$ . The BLT partitioning has sorted  $N_j$  before  $N_i$ .
  - b. For  $N_j$  it can be viewed as “used by” edge.
2. Parallelization: Search for all **source** blocks in the BLT graph, i.e. blocks which does not depend on other blocks (no outgoing needs edges) and collect them in layer  $L_i$ , which corresponds to equations or systems of equations that can be solved independently of each other.
  3. Delete all nodes in  $L_i$  from the block node graph and delete all edges to them.
  4. If there are still nodes in the graph, increase  $i$  by one and go to step 2.
  5. The sets  $L$  define a parallelization and the calculations are given starting from  $i=1$ .

We developed this algorithm in October 2009 and made a test implementation. However, our run-time infrastructure was not completely reentrant, so we were not able to make any simulation tests. Casella proposed in 2013 (Casella, 2013) a very similar algorithm but did not present any run-time speed-up results. However, in step 2, this algorithm searched for **sinks**, i.e. it was a dual algorithm.

Unfortunately, using the result of these basic algorithms for parallelization doesn’t give desired speed-up. The reason is that the obtained sections of a layer  $L_i$  are normally too small so overhead will give longer execution times. It is necessary to aggregate blocks into a section so the execution of a section outweighs the effort to set up and start the thread execution.

The approach to search for sources can be viewed as performing a calculation as soon it is possible. This approach can be taken bit further, if we consider the block nodes in the order defined by the BLT sorting and if we are collecting block nodes for  $L_i$  and a block node only has need edges from one section we add that block node to that section instead of deferring the calculation to  $L_{i+1}$ . The approach to search for sinks can on the other hand be viewed as performing the calculations just-in-time. We are then considering the nodes in the reverse order defined by the BLT sorting. If we are collecting block nodes for  $L_i$  and a block node only has a needed edges from one section we add that block node to that section instead of already doing the calculations in  $S_{i+1}$ . Please, recall that in this approach the sets are ordered in reverse calculation order.

In many cases the sets will include more parallel sections than we have cores. Thus, it is useful to merge sections within a set to make each section bigger to beat overhead. To get the sections balanced we need to have some estimate of the effort needed to perform the calculations of each block node. That is a difficult task for many reasons such as the solving of a non-linear problem may take different number of iterations. We have developed heuristic to get an estimate of the number of arithmetic operations needed to execute a block node. This includes also estimating the complexity of Modelica functions. Fortunately, we can make the result less sensitive to the estimates by generating code for more sections than we have cores. If we order the sections in estimated complexity order with the most complex first, the execution will start with them. If some estimates are bad or one nonlinear system needs more iteration, the other cores can be kept busy by handling remaining sections. Each section should have a minimum estimated effort.

From the two basic approaches discussed above indicates there may be a freedom in which parallel layer a calculation may be put in. To get fast translation it is not possible to check and evaluate all scheduling possibilities. To simplify the problem we focus on the most complex blocks and try to get as many of them in the same layers.

When the parallel sections of each layer have been identified, we need to be concerned with data. If data used and updated by different cores are adjacent, there is a risk of performance degradation due to *false sharing of cache lines*. Even if one core does not write the same double precision number as used by another core, the two variables might reside in the same cache line, typically 64 bytes of memory. We overcome this false sharing by adding padding, of typically 64 bytes, between the variables sets of the different sections.

### 3 Parallel Code for Equations

Consider the following model:

```

model ParallelCodeGeneration
  input Real u, U;
  Real x1, x2, x3;
  Real y1, y2, y3;
  Real X1, X2, X3;
  Real Y1, Y2, Y3;

equation
  der(x1) = f1(x1, u);
  y1 = g1(x1, u);

  der(x2) = f2(x2, y1);
  y2 = g2(x2, y1);

```

```

der(x3) = f3(x3, y2);
y3 = g3(x3, y2);

der(X1) = F1(X1, U);
Y1 = G1(X1);

der(X2) = F2(X2, Y1);
Y2 = G2(X2);

der(X3) = F3(X3, Y2);
Y3 = G3(X3);
end ParallelCodeGeneration;
}
// Sequence of calculations
der(x3) := f3(x3, y2);

// Result calculations
y3 := g3(x3, y2);
Y3 := G3(X3);

```

It consists of 3 dynamical systems with direct term (g1, g2, g3) coupled in series and 3 systems in series without direct term (G1, G2, G3).

The partitioning of the equations into layers of parallel sections as described above corresponds to a fork-join code generation scheme. OpenMP supports this scheme by using simple pragmas in the C-code. The following code will thus be generated by the proposed parallelization algorithm:

```

#pragma omp parallel
{
#pragma omp sections
{
#pragma omp section
{
der(x1) := f1(x1, u);
}
#pragma omp section
{
y1 := g1(x1, u);
}
#pragma omp section
{
der(X1) := F1(X1, U);
}
#pragma omp section
{
Y1 := G1(X1);
}
#pragma omp section
{
Y2 := G2(X2);
}
}
}
#pragma omp parallel
{
#pragma omp sections
{
#pragma omp section
{
der(x2) := f2(x2, y1);
}
#pragma omp section
{
y2 := g2(x2, y1);
}
#pragma omp section
{
der(X2) := F2(X2, Y1);
}
#pragma omp section
{
der(X3) := F3(X3, Y2);
}
}
}

```

Two parallel layers will thus be used followed by a sequence (of one block). The last two equations are not needed for solving the differential equations only to enable plotting.

Note that *no* mutual exclusion locks are needed since updating a variable is done only in one block, i.e. in one section and reading is always done in later layers, i.e. after resynchronization.. Between layers of parallel code, usual sequential code is used typically when potential parallel sections would be too small and sequential execution is faster.

### 4 Example: Electrical Circuit

Consider a Spice3 model of a 4-bit ripple-carry adder. The model is available in the Modelica Standard Library:

Modelica.Electrical.Spice3.Examples.  
Spice3BenchmarkFourBitBinaryAdder

The model adds two 4-bit numbers and is built out of four two-bit adders, Figure 1:

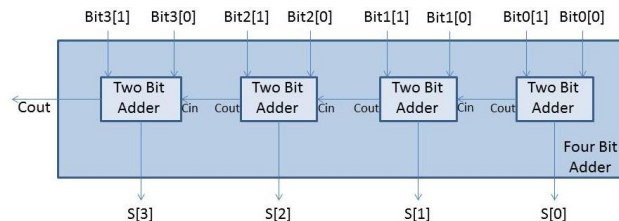


Figure 1: Four-bit adder

The two-bit adders are built out of two one-bit adders, Figure 2.

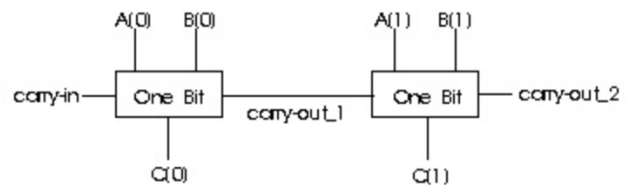


Figure 2: Two-bit adder

The one-bit adder is built out of nine NAND gates, Figure 3.



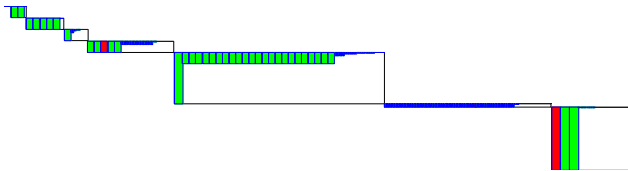


Figure 7: Parallel schedule using dual algorithm

The characteristics of this schedule are: pathLength = 60 697, speedUpFactor = 7.0, numberOfLayers = 15, numberOfCores = 325.

These schedules are not realistic for the target architectures with fewer cores available today. The granularity is too small since some of the sections just have a few operations. So merging of blocks must be done. Furthermore, it can be noted that some large blocks have ended up in different layers. In many cases, there is a freedom to move blocks between layers still not violating the data dependency. By appropriate heuristic rules, it is possible to achieve the schedule in Figure 8. In this case the maximum number of sections has been constrained to 4.

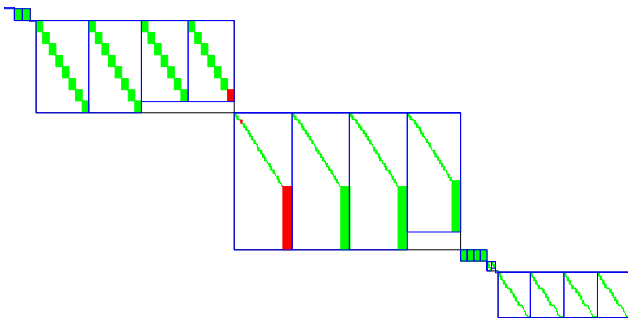


Figure 8: Parallel schedule with max 4 sections

The following characteristics is achieved: pathLength = 114 998, speedUpFactor = 3.7, numberOfLayers = 6, numberOfCores = 4.

The schedules presented are essentially Gantt charts with time advancing downwards. By using fixed width of the rectangles instead of width proportional to the number of unknown variables, we get Figure 9.

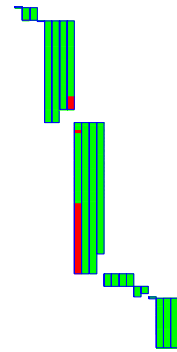


Figure 9: Gantt chart for 4 section schedule

The simulation time using this schedule is 16.5 seconds, i.e. a speedup of  $40.9/16.5 = 2.48$ . For the dual algorithm (first searching for sources), the simulation time becomes 18.0 seconds.

The utilization of the cores is quite low as can be seen in Figure 10:

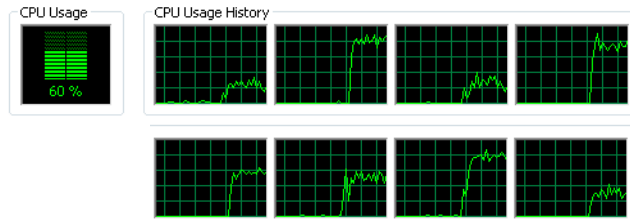


Figure 10: CPU usage with max 4 sections

It makes sense to increase the maximum number of sections to 8, 16 or even 32. The best simulation times achieved was 13.9 seconds, i.e. a **speed-up of 2.94**. The utilization of the cores then becomes 80%, Figure 11.

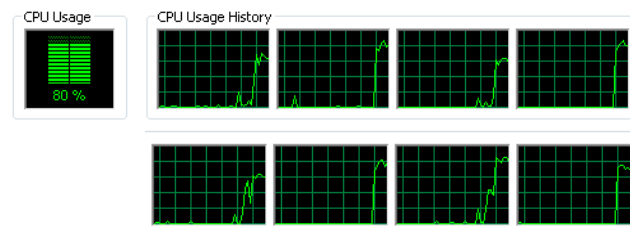


Figure 11: CPU usage with max 16 sections

The actual execution times are logged during simulation using high resolution timers and are presented as follows:

Name of block	Total CPU[s]	Min[us]
Seq 2 (71)	0.009	0.22
Par 3 [2] (59)	1.571	32.85
Seq 4 (45)	0.010	0.14
Par 5 [16] (98)	6.201	105.86
Par 6 [4] (428)	2.458	86.21
Par 7 [4] (49)	1.623	32.76
Par 8 [2] (36)	0.508	10.73
Seq 9 (20)	0.010	0.26
Par 10 [16] (87)	0.988	28.51

Seq  $i(m)$  means sequential layer  $i$  with  $m$  BLT blocks, i.e. no parallelization. Par  $i[n](m)$  means parallel layer  $i$  with  $n$  parallel sections and  $m$  BLT blocks. It is then seen that layer 5 has larger execution time than layer 6 contrary to the estimated times shown in the Gantt diagram. The reason is that layer 5 has 16 sections but only 4 cores are available, i.e. some of the sections must be executed sequentially.

## 5 Example: Evaporator

Consider simulation of a model of an evaporator included in the AirConditioning library as `AirConditioning.Examples.Evaporator` model and shown in Figure 12.

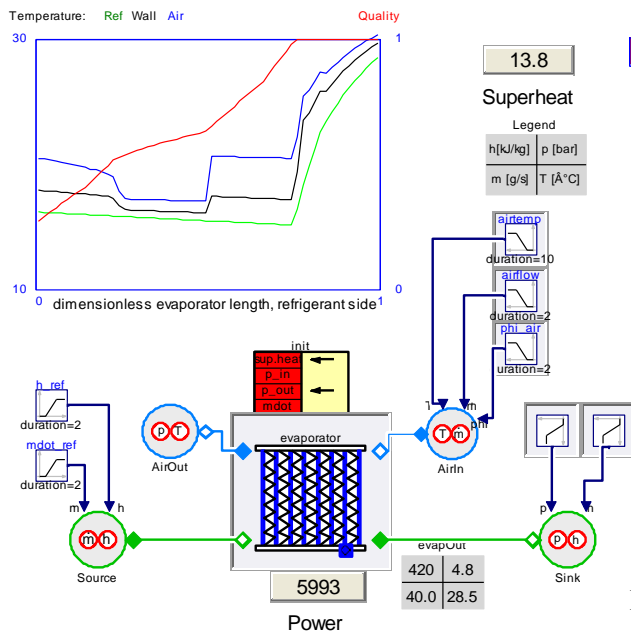


Figure 11: Evaporator model

The geometry of the evaporator for both air and refrigerant are discretized into segments. The example given in `AirConditioning` has `evaporator.n_segAir = 1` and `evaporator.n_segRef = 3`, which gives that the refrigerant is discretized into 18 cells along the flow direction. Each of these cells as well the sink has pressure and enthalpy as continuous states and there is additionally 18 continuous time states for the wall temperature giving a total of 56 states. The major computational task is to calculate the thermodynamic properties from the state variables. There are 9 non-linear systems of size 23 and 9 non-linear systems of size 21. The difference is due to the geometry of the heat exchanger. The tearing procedure results in problems with 3 iterations variables for all of them. If we set the stop time to 100 s, it takes 2.44 s to

simulate it on a normal laptop. We are here mainly interested in the speed-up when parallelize the calculation, so the absolute time is of less interest, however, it is of interest to consider the absolute time and number of calculations, because when the calculation are sufficiently short, the overhead to start and synchronize the parallel calculation the overhead will make the parallelized calculations take longer time than running then in sequence on one processor.

A causality analysis gives that we can first solve 9 of the systems in parallel followed by some other calculations and thirdly the remaining 9 systems.

Partitioning of the code with maximum of 16 sections can be represented in the graph in Figure 12. Green rectangles represent BLT blocks. These are merged into sections which are outlined in blue. The critical path is marked in red.

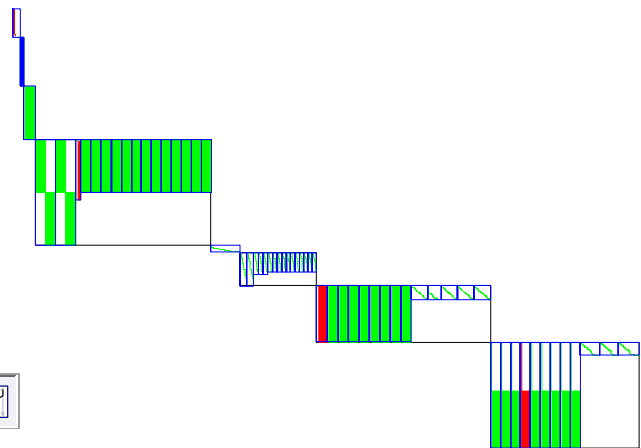


Figure 12: Schedule for Evaporator using max 16 cores

It takes about 45  $\mu$ s to solve such a nonlinear system of equations. **With 4 cores/8 threads we get a speed-up factor of 2.1.**

Let us make the model more complex by making a finer and more desirable discretization by setting `evaporator.n_segAir = 5` and `evaporator.n_segRef = 10`, which gives that the refrigerant is discretized into 60 cells along the flow direction. Each of these cells as well the sink has pressure and enthalpy as continuous states and there is another 60 continuous time states for the wall temperature giving a total of 182 states. There are now 30 systems of size 112 and 30 systems of size 110. The tearing procedure results in problems with 11 iterations variables for all of them. The simulation time becomes now 58.8 s, compared to 3.7 s for when using a less fine discretization. A causality analysis gives that we can first solve 30 of the system in parallel followed by some other calculations and thirdly the remaining 30 systems can be solved in parallel. It takes 60-90  $\mu$ s to solve such a system. We can parallelize these computations. **With**



**4 cores/8 threads the factor is 3.4.** The calculations between the systems of equations can be split into two parts that can be run in parallel. One part takes  $2.2 \mu\text{s}$  to execute and the other takes  $6.7 \mu\text{s}$ , so running them in series takes  $8.9 \mu\text{s}$ . If we run them in parallel it takes  $23 \mu\text{s}$ , due to the overhead to start up and synchronize parallel threads.

## 6 Example: Multi-Body Systems

Let us consider simulation of multi-body systems. A major task is to invert the mass-matrix to solve for the accelerations. If the system has kinematic loops there are additional non-linear and linear systems. The factorization of a Jacobian is a major task when solving a system of equations. However, when parallelizing that, it gave faster calculations only if the size of the Jacobian was greater than  $300 \times 300$ .

Another major task is calculation of forces and accelerations. For a tree structured mechanism, there are obvious possibilities to parallelize each branch of the tree into different threads. Such possibilities will be investigated in the future.

### 6.1 Real-Time Simulation and Inline Integration

What has been discussed above applies to hardware-in-the-loop simulation (HILS) as well as when variable step-size integration methods are used.

For HILS of stiff models such as multibody systems including bushings, implicit methods are needed. Inline integration (Elmqvist, et.al., 2002) can then be used. It means that the discretization formulas are merged with the model equations and the entire merged set of equations is symbolically transformed.

The parallelization method can be used on the merged set of equations. However, implicit methods typically give one large system of non-linear equations to solve, i.e. the above method is not useful as described.

To handle this situation, we have introduced a **decouple operator** which delays the signals one step corresponding to changing from an implicit relationship to an explicit. The result is that the large system of equations decomposes into smaller systems that both are faster to solve and that can be solved in parallel.

Using this technique, we have been successfully simulated vehicles with about 150 DOFs (Degrees-of-Freedom) in real-time using a 1 ms step size and implicit Euler. By decoupling the front and rear wheel suspensions respectively from the chassis

(having large mass) and by decoupling each wheel from the wheel suspensions we achieved a speed-up of 1.5. The two decoupling elements decoupling front and rear suspensions from the chassis can be seen in the VDL (Vehicle Dynamics Library) diagram in Figure 13:

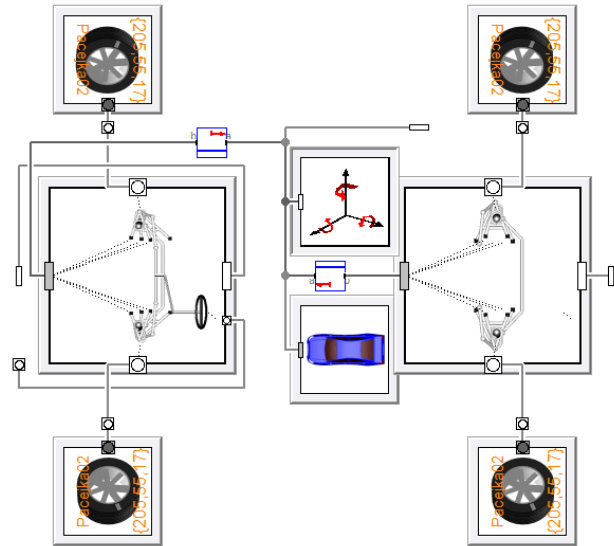


Figure 13: Decoupling of VDL model

Parallelizing the resulting smaller non-linear systems of equations gave a **further speed-up of 2.8** with 4 cores/8 threads.

## 7 Conclusions

Modelica is well suited for automatic parallelization of equation execution using many cores. We have described a technique for such automatic partitioning. The obtained results are good for models from various domains. For a thermo-dynamic model a speed-up of 3.4 was achieved, for electrical 2.9 and for mechanical HILS 2.8 using 4 cores/8 threads.

## References

- Aronsson, P., and P. Fritzson (2002). **Multiprocessor Scheduling of Simulation Code from Modelica Models**. In *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, Mar. 18–19, 2002.
- Aronsson P. (2006): **Automatic Parallelization of Equation-Based Simulation Programs**. *Institutionen för datavetenskap, 2006*.

- Casella, F. (2013): **A Strategy for Parallel Simulation of Declarative Object-Oriented Models of Generalized Physical Networks**. *5th International Workshop of Equation-Based Object-Oriented Modeling Languages and Tools*, April 2013, University of Nottingham, UK. Linköping University Electronic Press.  
[http://www.ep.liu.se/ecp\\_home/index.en.aspx?issue=084](http://www.ep.liu.se/ecp_home/index.en.aspx?issue=084).
- Elmqvist H., Mattsson S.E., Olsson H. (2002): **New Methods for Hardware-in-the-Loop Simulation of Stiff Models**, *2nd International Modelica Conference*, DLR, Oberpfaffenhofen, Germany, March 18-19.

# A toolchain for Rapid Control Prototyping using Rexroth controllers and open source software

Nils Menager   Niklas Worschech   Lars Mikelsons  
Bosch Rexroth AG  
Rexrothstr. 3, 97816 Lohr a. Main

## Abstract

Taking a look at project costs from a financial point of view, the commissioning times of new industrial systems become more and more important, as they significantly drive the costs. Hence, the reduction of commissioning times is part of current research. Besides the use of simulation and the coupling between hardware and software (Hardware-In-The-Loop-Simulations), *Rapid Control Prototyping* offers a huge potential to reduce commissioning times. Until now, most of the toolchains use special hardware in combination with commercial simulation software, which leads to some serious drawbacks. In this work, a toolchain for Rapid Control Prototyping using industrial controllers and open source software is presented.

*Keywords:* *Rapid Control Prototyping; Modelica; OpenModelica; Hardware-In-The-Loop; Bosch Rexroth; real-time simulation*

## 1 Introduction

### 1.1 Motivation

According to the V-model of mechatronic product development (VDI 2206, [1]), the first step during the design of a new industrial system, after a rough pre-calculation of the required components, is to set up a simulation model of the system under consideration. Therefore, at Bosch Rexroth, our in-house tool *Rexroth Simster* is used. *Rexroth Simster* is a simulation environment, which allows object-oriented modelling of technical, mainly mechanical, electrical and hydraulic systems. Clearly, the tool includes powerful numerical solver to perform the simulation. After suitable components and parameters are determined inside the simulation environment, the controller concept has to be tested using a real hardware controller. At this point, until now, the simulation model of the controller

is not used any further and the controller is set up from scratch inside the development environment of the controller. At Rexroth, mostly *IndraWorks* as standard tool for the development of controller algorithms and the design of the whole controller is used. In many cases, even the developed controller structure inside the simulation tool is not used anymore. Instead, pre-defined standard control algorithms are used.

However, it is clearly desirable to adapt the complete control algorithm from the simulation environment. There are already possibilities to use virtually designed control algorithms on real-time operating systems using Hardware-In-The-Loop-Setups. Two possibilities are using a dSPACE [2] system or a xPC system in combination with *Matlab/Simulink* [3]. Though, using such a toolchain leads to three serious drawbacks. First, these systems are very expensive. Second, even in that case the control algorithm has to be re-implemented on the real control hardware after testing on the real-time system. Furthermore, it has to be taken into account that the usage of such commercial real-time systems leads to dependencies to external software (e.g. *Matlab/Simulink*). Clearly, such a dependency for the generation of code for the PLC (*Programmable Logic Controller*) is not desirable. Moreover, if new features should be implemented, this is a big disadvantage, because the code generation of the commercial tools can not be modified.

The aim of this work is to set up a toolchain for Rapid Control Prototyping with a Rexroth controller (IndraControl L45) using open source software and Modelica for the modeling part, i.e. a toolchain, which is completely independent from external software and hardware. To be more precise, this toolchain enables the engineer to transfer virtual controller models modeled in Modelica to standard Rexroth controllers. To validate the functionality, in this contribution, the controller is used in a Hardware-In-The-Loop setup. The

validation of the controller in combination with a real system is part of current work.

## 1.2 Outline of this paper

In the first section of this paper, a short introduction into Rapid Control Prototyping is given. Furthermore, a standard toolchain for using Rapid Control Prototyping nowadays is discussed. In the second section, after requirements for the toolchain have been defined, the specific parts of the toolchain are presented. The functionality of the toolchain is finally verified using a Hardware-In-The-Loop setup consisting of a real Rexroth hardware controller running controller code of an industrial control algorithm created in Modelica and a model of a hydro-mechanic system. In the last part, a short summary and an outlook on further investigation is presented.

## 2 Rapid Control Prototyping

Rapid Control Prototyping is a computer-aided method for developing and testing control algorithms quickly on real-time operating systems. This approach allows to investigate how the control algorithm will behave later on the real hardware controller. Rapid Control Prototyping includes all steps between the definition of the controller specifications and the implementation of the final control algorithm. The single steps during the Rapid Control Prototyping process are shown in the V-model in figure 1. The left part of the V-model shows the way from the specification of the requirements to the implementation of the controller,

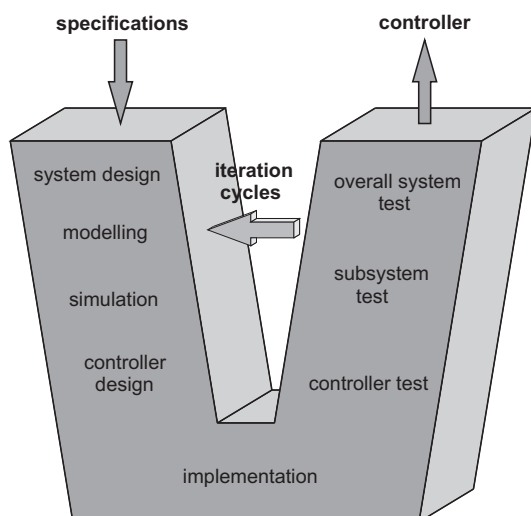


Figure 1: V-model showing the Rapid Control Prototyping process

in the right part the functionality of the implemented algorithm is verified. The functionality of the algorithm is then compared to the requirements specified at the beginning of the cycle. If there are differences between the required specifications and the actual functionality, another iteration cycle is necessary. This procedure is repeated until the actual behavior of the controller and the specifications fit together.

### 2.1 dSPACE/Matlab

Nowadays, the usage of Rapid Control Prototyping is already standard in different industry branches, e.g. the automotive industry. Therefore, for example dSPACE (*Digital Signal Processing And Control Engineering*) systems can be used. On a dSPACE box a real-time operating system is working which allows to execute code in real-time on the device. To generate the code, e.g. special toolboxes from *Matlab* can be used. These toolboxes allow to generate executable code for the dSPACE box in a very short time. Using this toolchain it is possible to develop and test control structures on a real-time operating system in an easy and fast way.

Although this method offers the possibility to test algorithms quickly, it does not avoid the re-implementation on the final hardware device, which is a big disadvantage. Besides the fact, that a re-implementation is time consuming and a possible error source, the portability is potentially incalculable. It cannot be guaranteed that the controller on the final target behaves in the same way as the controller on the test device. Other disadvantages are the costs of such a dSPACE system and the dependencies to the commercial software.

Because the *Matlab Code Generation* only works for special operating systems and special hardware, there are no direct possibilities to adapt the code generation for other devices like the Rexroth controller. Of course, it could be tried to wrap the code could for the use on other devices, but even in this case, this code has to be compiled for the target operating system. If only one single part of the code cannot be compiled for the operating system, it is impossible to execute the code on the hardware. Hence, it is necessary to develop a toolchain, which is open source and therefore applicable to different hardware devices. The development of such an open source toolchain is the topic of this work.

### 3 Realization of the toolchain

For the realization of the toolchain, clearly, different tools are needed. The structure of the toolchain is shown in figure 2. The starting point is the simulation environment *Rexroth Simster*. As already said in the introduction, the first step during the development of a new system is to set up a simulation model of the system and the controller inside the simulation environment. The simulation tool *Rexroth Simster* will, in one of the next releases, support both the usage of models from an internal library (written in C/C++) and of Modelica models. It makes sense to use Modelica models, because they offer three big advantages. The first one is, that Modelica models are easy to create and support an object-oriented modeling structure, either using a graphical user interface or the Modelica editor. Second, in order to simulate Modelica models, they have to be compiled to C/C++. There exist both commercial (e.g. *Dymola*) as well as open source (e.g. *OpenModelica*, *JModelica*) Modelica-compilers. In this work, of course the open source *OpenModelica*-compiler (OMC) is used. Furthermore, Modelica is a widespread language, which is used frequently in the industry.

The controller model shall then directly be used on the real PLC. The PLC used here is a common Rexroth controller (Rexroth IndraControl L45). Therefore it is necessary to compile the model of the controller for the operating system of the hardware. The real-time operating system running on the hardware is *VxWorks*. To execute the code and run the simulation on the PLC, a simulation core, which runs and manages the simulation, is additionally required. This simulation runtime has also to be compiled for the operating system *VxWorks*. To compile the system and the runtime and load the compiled libraries onto the hardware, the development environment *WindRiver Workbench* is used. This environment includes among other things a *VxWorks* compiler. The simulation of the controller model can then be executed in parallel to the main thread on the PLC. This first step, to compile and load the model on the PLC, is shown via the grey, dashed arrow in figure 2.

To allow the exchange of data between the PLC and the simulation environment during a HiL-simulation, an interface between the newly created thread running the controller code (controller thread) and the original *IndraWorks* thread (main thread) is required. To get access to the Rexroth IndraControl L45, the soft-

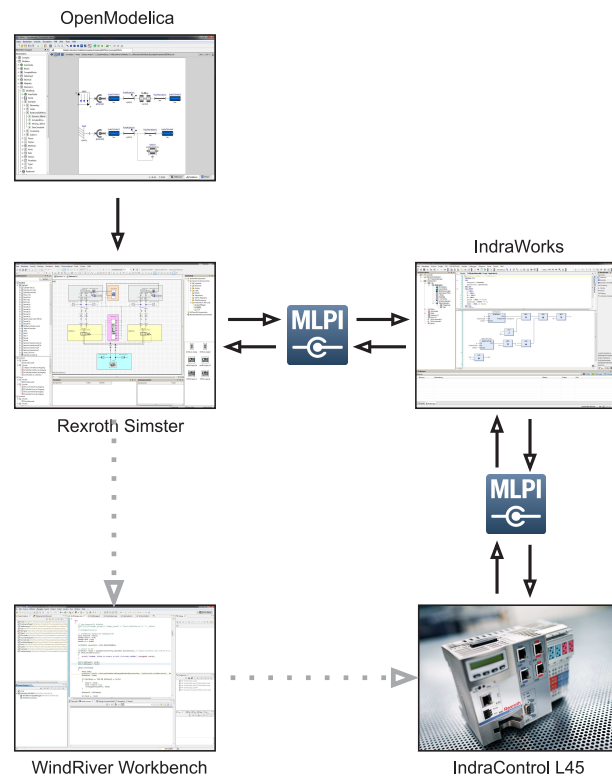


Figure 2: Structure of the toolchain

ware *IndraWorks* is used. *IndraWorks* is a standard tool for the development of control algorithms and the design of Rexroth PLCs. The connection between the controller thread and the main thread is realized with a function block according to IEC 61131 [4] inside the *IndraWorks*-application, which is used as an interface. In order to run the HiL-simulation, data has to be exchanged between the PLC and *Rexroth Simster*. Therefore, the MLPI (*Motion Logic Programming Interface*) is used [5]. The MLPI is a programming interface for high level programming languages (C/C++/C#/VBA/Java/LabVIEW/...). It can be used to write applications, which can be used to configure and run a Bosch Rexroth controller which supports the MLPI interface technology, like the IndraControl L45.

#### 3.1 Used software components

In the following sections, a short overview and more detailed information about the tools used in this work are given.

##### 3.1.1 Rexroth Simster

The simulation environment *Rexroth Simster* is an in-house tool developed by Bosch Rexroth. It covers multiple domains (mechanic, hydraulic, electric) and has

been developed for the design and optimization of controlled automation systems. The component library includes both generic and Rexroth specific components, which can simply be placed on the worksheet using drag and drop. *Rexroth Simster* includes a special simulation core. This simulation core offers an interface for models from the standard *Simster* libraries, which are implemented in C/C++. The C/C++ code generated by the *OpenModelica*-compiler implements the same interface. Thus, the simulation core can handle both models from the own internal library and Modelica models. Detailed information about the developed simulation runtime is given in [6].

### 3.1.2 WindRiver Workbench

*WindRiver Workbench* is an eclipse-based development environment for *VxWorks* and is used in version 3.3. *VxWorks* is a real-time operating system, which is mainly used in embedded systems and is the operating system running on Rexroth hardware controllers. Included in the development environment is a *VxWorks* compiler, which generates executable code from C/C++-code. The tool is used to compile both the controller model and the simulation core for the use on the hardware controller. The classes inside the simulation core are compiled into dynamically linked libraries, which leads to *.out*-files. These *.out*-files have then to be moved to the internal flash card of the Rexroth hardware. This is done using an FTP client.

### 3.1.3 IndraWorks

*IndraWorks* is a tool developed by Bosch Rexroth and is used as standard tool for the development of control algorithms and the entire configuration of the PLC. Inside *IndraWorks*, an application to run on the hardware can be created. After having configured the connection parameters (IP address, type of connection), the algorithms are developed using the IEC 61131-3 standard PLC programming languages. Furthermore, additional languages especially for the use of motion commands (PLCOpen) are available. During the runtime of the controller, the process can be visualized and monitored using plotter and other visualization tools.

### 3.1.4 Motion Logic Programming Interface

The *Motion Logic Programming Interface* is an interface supporting many high level programming interface and is also developed by Bosch Rexroth. Using

this interface, it is possible to write applications to configure and run Bosch Rexroth devices which support the MLPI interface technology. It contains a set of headers and libraries. There are 8 different libraries, which allow access to different parts of the controller:

- *mlpiAPILib* includes functions to connect and disconnect MLPI
- *mlpiSystemLib* includes functions to read system information like temperature, diagnosis data and the firmware version
- *mlpiParameterLib* includes functions to read/write parameter
- *mlpiLogicLib* includes functions to start/stop/reset the PLC, load PLC programs, browse/read/write symbol variables
- *mlpiMotionLib* allows access to general motion functions, single axis motion, cyclic commands and synchron axis motion
- *mlpiContainerLib* allows cyclic read/write access with fast container buffer mechanism
- *mlpiWatchdogLib* includes functions to monitor the user application
- *mlpiTraceLib* includes functions for the trace configuration and to add, collect and view debug information.

There are four different MLPI toolboxes, each supporting a different programming language. Here, the toolbox for C/C++ is used. In this work, MLPI is used on the hand side to realize the data exchange between the simulation tool *Rexroth Simster* and the hardware controller. Furthermore, MLPI is used as interface between the user and the controller to change controller parameter inside the controller code. The structure and functional principle of MLPI is explained in [5].

## 3.2 Connecting the different components to the toolchain

To ensure the functionality of the toolchain, some additional aspects must be considered while connecting the different parts to the toolchain. The different aspects are discussed in the following, each in an own subsection.

### 3.2.1 Modifications inside the simulation core

An important point is the library handling in *VxWorks*. Hence, each location inside the code loading a library has to be modified. To ensure the functionality in both the new operating system (*VxWorks*) and the old environment (*Microsoft Windows*), pre-processor commands are used to decide which implementation is used. Using this method the same runtime can be used in both *Rexroth Simster* and on the PLC, which is an important requirement. To load dynamic libraries in *VxWorks*, the following basic framework has to be used:

```
int libraryFile = open("lib.out", O_RDONLY, 0777);
if (libraryFile == ERROR)
    // Error loading library

MODULE_ID c_moduleId = loadModule(libraryFile ,
    LOAD_ALL_SYMBOLS);

close(libraryFile);
if (c_moduleId == NULL)
    // Unable to load as module

extern SYMTAB_ID sysSymTbl;
SYM_TYPE symType;
double (*funcPtr)(int);

if (symFindByName(sysSymTbl, "name",
    (char**) &funcPtr, &symType) == ERROR)
    // Symbol not found

double a = funcPtr(2);
```

After the library is loaded by the *open* command, all symbols are loaded using the *loadModule* function. This allows to get access to the functions inside the library. The next step is to create a function pointer. The function pointer in this example points on a function, which gets an integer as input variable and which returns a double value. The last step is to search a specific function in the symbol list using the *symFindByName* function. This function pointer can then be used to call the function inside the library.

### 3.2.2 Synchronization of the HiL-setup

The next aspect is the synchronization between the simulation of the system inside *Rexroth Simster* and the code execution on the PLC. It is clear, that both processes have to run synchronized, so that the exchanged data fit together. *Rexroth Simster* is a windows application and therefore, without any modifications, not real-time capable. That means, that the calculation time depends on the complexity of the model and the workload of the used operating system. Thus, the simulation can be faster or slower than real-time. In contrast, a PLC is a hard real-time system with a fixed cycle time. At the beginning of one cycle, the

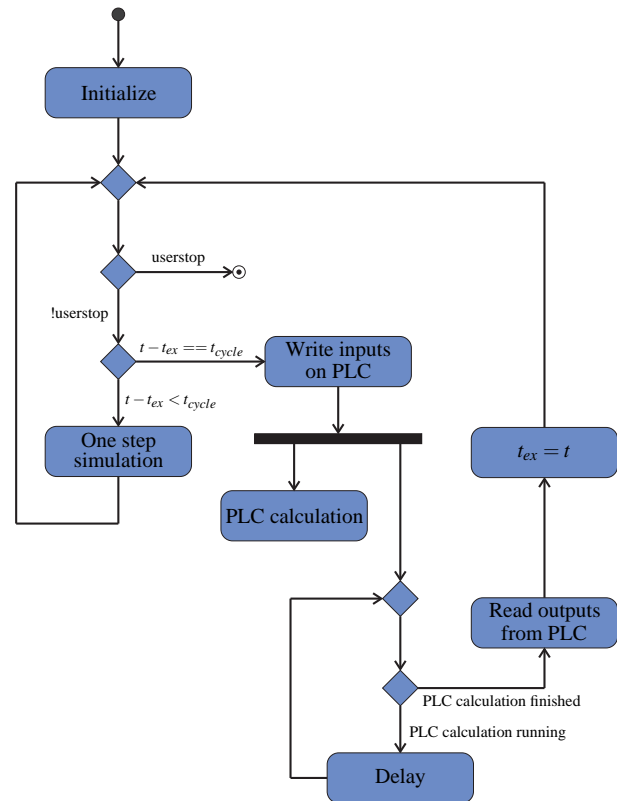


Figure 3: UML Diagram showing the synchronization

inputs of the controller are read. Then, the control algorithm is executed and the output data based on the input data is computed. The last step is to write the calculated values to the output. Therefore, the controller expects input signals in real-time. Using a standard PLC containing a standard *IndraWorks* application (i.e. the controller computes the output in real-time) in a Hardware-In-The-Loop-setup with the *Rexroth Simster*, there are two possibilities: either the simulation is forced to run in real-time or the controller has to be adapted to the simulation speed of the simulation environment.

Because the second way has some big advantages, the slowdown of the controller has been realized. One main advantage is the user-friendliness. Main users of this toolchain are engineers like start-up engineers, who shall design a new industrial system and do not have the possibility to use a real-time operating system on their working computer. Additionally, this way is more comfortable, as there are no limits with regard to applicable numerical solve algorithms or the complexity of the model.

To realize the slowdown of the controller, a trigger variable inside the controller application to start the task is necessary, which activates one calculation step

on the controller. After the simulation is progressed by the length of the cycle time, the simulation is stopped and the values of the input variables on the controller are set inside the *IndraWorks* application using MLPI commands. After that, the trigger variable is set to *true*, which starts the calculation of one single step of the controller. At the end of the computation, another variable, which indicates that the computation is finished, is set to *true*. The simulation environment reads the output values from the controller and sets both the trigger variable and the variable indicating the end of the calculation to *false*. These steps are repeated until the end of the simulation time is reached.

In the setup described in this contribution, however, the control algorithm is not implemented in *IndraWorks* and therefore not computed inside the main thread of the controller, but in the separate controller thread running in parallel to the main thread, which makes the situation more complicated. This means, that the program is not controlled via an *IndraWorks* task like in the case discussed before.

### 3.2.3 Establishment of a connection between the different threads

The next challenge is the establishment of the connection between the controller thread and the main thread, because the MLPI commands allow only access to variables and functions inside the *IndraWorks* application running on the PLC. The connection between both threads is realized via a function block inside the *IndraWorks* application. It is possible to link an external implementation to a function block, so this function block has no own implementation.

To ensure that the application will find the missing function implementations, the external implemented functions have to be registered using the MLPI function *mlpiLogicPouExtensionRegister* from the *mlpi-LogicLib*. This function provides the possibility of using C/C++ extensions within the IEC 61131-3 environment *IndraWorks* and describes the mapping between the function block name in *IndraWorks* and the function name in the C/C++ implementation. The variables that shall be exchanged can now be defined as variables inside the function block in the main thread. Then, both the simulation inside *Rexroth Simster* and the controller on the PLC can get access to the variables using MLPI functions as well as read and write the variables. The structure of the communication between the two parallel threads is shown in figure 4.

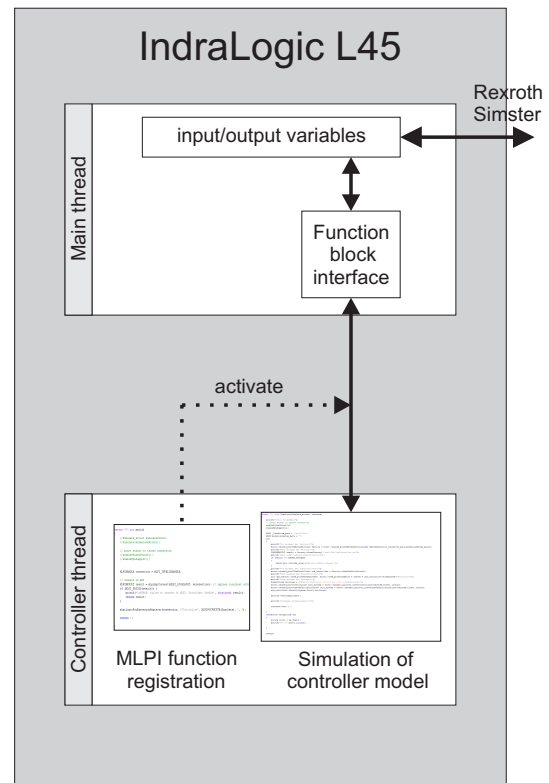


Figure 4: Interface between main thread and controller thread on the hardware controller

After having established the connection between the two threads, the synchronization between the simulation of the system inside *Rexroth Simster* and the controller can be realized analogous to the technique described before. The trigger variable is defined inside the *IndraWorks* application and can be set by both the simulation environment and the controller thread.

### 3.2.4 Initialization of the toolchain

The next aspect to be considered is the initialization of the code execution on the controller. After the compilation of the simulation core and the controller code, all dynamic libraries are available on the internal memory. To start the controller, a main function to manage the code execution (load the libraries in the correct order, call the functions to initialize the solver and the system, start the code execution) is necessary. This function has to be executed before the controller starts, so that all libraries are loaded and all instances of the classes are already initialized. This is the function later called automatically via the function block interface.



### 3.2.5 Resulting workflow for the toolchain

Regarding all the aspects discussed before, the resulting workflow for the toolchain can be derived. Assuming, that the simulation model of the system and the controller inside the simulation environment *Rexroth Simster* already exist, the first step is to compile the simulation core and the controller code for the operating system of the controller (*VxWorks*). Therefore, a new project inside WindRiver Workbench has to be created. The classes inside the simulation core have to be compiled into dynamically linked libraries, which leads to *.out*-files. These *.out*-files have then to be moved to the internal flash card of the Rexroth hardware. This is done using an FTP client. The next step is to register the executable main function containing the initialization of the simulation on the controller (see section 3.2.4). The registration is done using the MLPI function *mlpiPouExtensionRegister*. The syntax is as follows:

```
MLPIRESULT mlpiLogicPouExtensionRegister (
    const MLPIHANDLE connection ,
    const WCHAR16* name ,
    const MLPIPOUFCPTR function ,
    const ULONG signature = 0 ,
    const ULONG version = 0).
```

The first input parameter is the connection handle automatically created when a connection to the hardware via MLPI is established, the second parameter is the name of the POU (Program Organization Unit) in *IndraWorks*, in this case the name of the function block, the third parameter is the function pointer to the C/C++ implementation, while the fourth and fifth describe the signature of the POU interface and the version of the POU library, if implemented within a library, and have not necessarily to be set, as they are predefined with 0 [7].

Now the implementation of the function block interface inside the *IndraWorks* application is made known to the *IndraWorks* application. As the next step, the *IndraWorks* application, which only consists of the function block with the external implementation and the definitions of the variables to be exchanged during the simulation as well as the trigger variable (see section 3.2.2), can also be uploaded to the hardware device (if the *IndraWorks* application is uploaded before the registration of the functions is executed, there will be linker errors for the external implementation of the function block).

The next step is to start the initialization of the controller on the hardware, i.e. to active the func-

tion block. Therefore the task controlling the function block has to be started. This can again be realized via MLPI. The task is defined as *triggered task*, which allows to start the task setting the activation variable to *true*. The start of the main function effects, that all necessary libraries are loaded and the simulation manager is started. Inside the simulation manager, a query is continuously performed, whether the trigger variable to start one calculation step is set or not. For the connection to the hardware device from the *Rexroth Simster* side, a special component is necessary, which has been developed for HiL-tasks (MLPI-Coupler). The component has several inputs and outputs and contains the MLPI-commands to both write the data from the different inputs on the device and read the data from the device and set the values to the outputs of the component. The names of the variables inside *IndraWorks* can be set as component parameters.

The last step is to start the simulation inside *Rexroth Simster*. The cycle time between the exchange of the data can also be set in the MLPICoupler component. The simulation of the system triggers then the simulation on the hardware device. The synchronization between both simulations is realized as described in section 3.2.2.

### 3.2.6 Automation of the toolchain

The toolchain presented in this work is not fully automated until now. In one of the next releases of *Rexroth Simster*, Modelica support will be added to the simulation environment. Until now, an additional Modelica environment is necessary to build up the controller simulation model. In the future, Modelica models can directly be created inside the simulation tool. The code generation is integrated into the simulation core, so that the executable code can directly be generated. This allows a fully automated toolchain, where the controller model can be set up inside the simulation environment and automatically be compiled and sent to the controller. Starting the simulation inside *Rexroth Simster* activates the toolchain (compile the controller model, transfer the code to the controller, load the simulation core libraries, start controller code execution).

## 4 Application on an example system

To verify the functionality, the toolchain is used to develop an appropriate control structure for the control

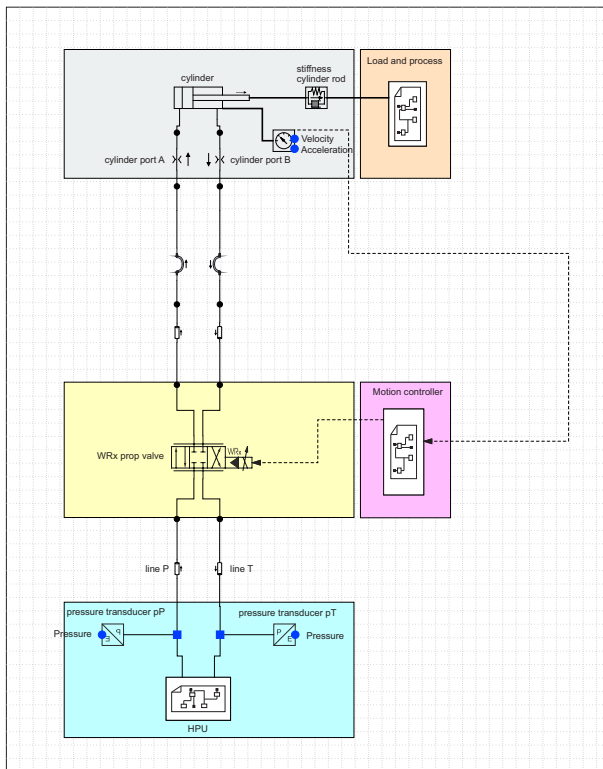


Figure 5: System model inside Rexroth Simster

of an industrial hydro-mechanical system, namely a single hydraulic axis. This system model is built inside *Rexroth Simster* and simulates the behavior of the hydro-mechanical system. The structure of this simulation model can be seen in figure 5. This model also includes a control loop which has been developed virtually inside the simulation environment.

#### 4.1 Model of the system

The single axis model consists of five sub-blocks, each highlighted in a different color. The light blue block is the HPU (hydraulic pumping unit), i.e. it realizes the oil supply for the hydraulic system and includes the oil tank, a variable pump which is powered by an electric motor. The motor speed is power- and pressure-controlled. Additionally an accumulator is included in the HPU to ensure the oil supply for temporary high demands on oil.

The yellow block shows a generic WRx proportional valve, which limits the volume flow of the hydraulic fluid. Using the input signal port of the valve, the spool position of the valve can be modified. The valve's dynamics is modelled with a PT2-behavior with power limit, the flow is modelled via a characteristic curve depending on the piston stroke ( $Q = f(s)$ ).

The connections between the pump and the valve are modelled by lines including frictional losses. The differential cylinder is modelled inside the grey box. The simulation model of the cylinder considers Stribeck friction (static friction, running friction and Coulomb friction), internal leakage and external leakage. Additionally the cylinder model has two end stops for the piston, which are implemented using momentum conservation (optionally a coefficient of restitution can be specified).

The load is modeled inside the orange sub-box and considers the force resulting from the load mass, the gravity force, the plastic and the elastic deformation. The velocity of the cylinder piston is defined by the user and is available as characteristic curve in the form  $v = f(t)$ . The position profile for the cylinder, which can be obtained through integration of the velocity profile, is shown in figure 6 (blue curve).

#### 4.2 Modeling the controller

To realize the control, the current position of the cylinder piston has to be measured. Therefore, the internal position measuring system of the cylinder, which is included in the cylinder model, is used. As first try for the controller structure a position controller is used. The position controller compares the current position of the piston with the desired position from the profile. The difference (control error) is then multiplied by a gain factor (P-controller). The profile and the controller structure are implemented inside the rose box in figure 5.

#### 4.3 Starting the RCP process

To transfer this controller model to the real hardware device, the toolchain which has been explained in section 3 is used. As numerical solver, the explicit Euler algorithm is used. Note, that the controller model contains an ODE from the integrator component to calculate the position from the velocity profile. For the verification, the results of this HiL-simulation using the developed toolchain are compared with the results produced by a simulation of both the system and the controller inside *Rexroth Simster*. Both results are shown in figure 6, the complete simulation inside *Rexroth Simster* in red, the simulation using the Rapid Control Prototyping toolchain in green.

It can be seen that both curves are very similar, but of course not identical. This is because the controller

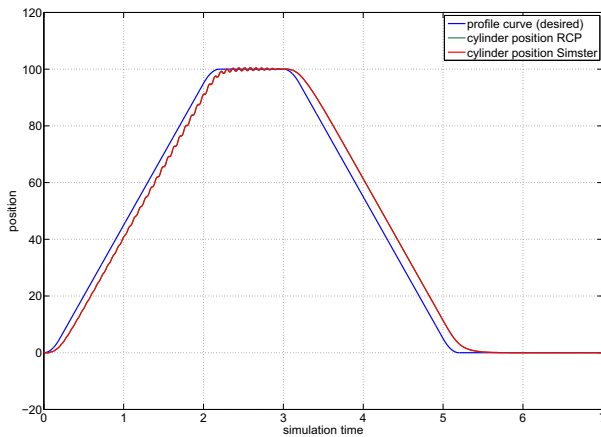


Figure 6: Results from the first controller implementation and comparison with the Rexroth Simster simulation

inside *Rexroth Simster* is a continuous controller and gets the position update from the cylinder in every solver time step, while the simulation on the hardware device (like every hardware controller) is updated only in the cycle time of the connection between simulation and hardware. But as the differences are very small, the functionality of the toolchain and the data exchange can definitely be verified.

#### 4.4 Improvements on the controller

If again the V-model of the controller development in figure 1 is considered, the first cycle is now finished. But, if the current result is compared to the desired result, another iteration cycle due to the existing oscillations is necessary. It is clear, that a simple P-controller cannot fulfill the control task. In the second iteration, a velocity feed forward to minimize the gap and an additional control part to minimize the oscillations is integrated into the controller. Therefore, the Modelica code is modified inside the simulation environment. After suitable parameters are determined, the controller structure is again transferred to the IndraControl L45 controller using the toolchain to investigate the functionality on the hardware. Figure 7 shows the result of the improvement of the control algorithm.

Taking a look on the results after this iteration cycle, it can be seen, that the oscillations could be removed. It can be assumed, that the developed controller structure, in general, is suitable to solve the control task in this example (the desired positions are reached without oscillations). In the practice, one or two additional iteration cycle would be performed in order to tune the

parameters to maybe get an even better parameter set, that shifts the current position more towards the desired position to minimize the gap. However, this is skipped at this point.

## 5 Summary and outlook on further investigations

In this work, a toolchain for Rapid Control Prototyping using an industrial Rexroth hardware controller based on open source software is presented. This toolchain allows to reduce commissioning times, avoiding the re-implementation of the controller structure from the simulation environment inside the development environment of the hardware controller. Furthermore, the toolchain is based on open source software. This ensures, that the functionality is independent from software developed by external companies, i.e. additional features can easily be implemented.

The functionality of the toolchain is also verified with an example. Here, the big advantages of Rapid Control Prototyping get visible. The control structure is developed and pre-tested easily inside the simulation environment. To test this control algorithm on the real hardware, until now, it was necessary to re-implement the algorithm. Using the developed toolchain, the re-implementation is no longer necessary, because the final hardware, which is applied later on the real system, is used for the Rapid Control Prototyping Process. The development process consists of several iteration cycles (see 1). In this example, we used three iteration cycles (third one not explicitly shown here), hence, three re-implementations could be saved. In

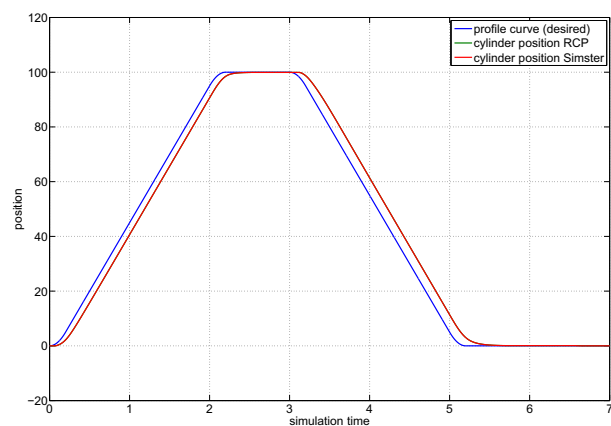


Figure 7: Results from the second controller implementation and comparison with the Rexroth Simster simulation

more complex systems, of course, the controller structure also gets more complex.

In this contribution, the RCP toolchain is verified virtually in a HiL-setup. In the future, the controller executing the code has to be tested on a real system. Therefore, the *IndraWorks* program has to be adapted. In the HiL-setup, the simulation inside *Rexroth Simster* uses MLPI commands to write the values to the variables in the *IndraWorks* application. If the controller is connected to a real system, the input and output signals are transferred via the input and output ports on the controller. Additional code for the mapping between the I/O ports and the variables is necessary. An important point is the observation of the calculation times. It has to be investigated in the future, how the strict observance of the calculation times can be guaranteed.

Another part of the future work is to fully automate the toolchain, as described in section 3.2.6.

The simulation core can not only be used to simulate controller models to realize Rapid Control Prototyping and couple hardware and software in a Hardware-In-The-Loop simulation. It is also possible to simulate whole system models, which opens the door to many other fields of application. One field of application are alternative control concepts like Model Predictive Control. Model Predictive Control calculates the current control action by solving an optimal control problem at each sampling instant using the current state of the plant as initial state. The cost function of an optimal control problem is optimized subject to different constraints. One main constraint is the system dynamics in the form  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ . This condition requires the simulation of the system in each optimization step.

## References

- [1] Richtlinie, V. D. I. "2206" - Entwicklungsmethodik für mechatronische Systeme, Berlin 2004
- [2] <http://www.dspace.com/en/pub/start.cfm>
- [3] <http://www.mathworks.com>
- [4] John, K.H.; Tiegelkamp, M. - IEC 61131-3: Programming Industrial Automation Systems, 2010
- [5] Engels, E.; Gabler, T. - Universelle Programmierschnittstelle für Motion-Logic Systeme - Struktur, Funktionen und Anwendung in Forschung und Lehre, Tagungsband AALE 2012
- [6] Worschech, N.; Mikelsons, L. - A Toolchain for Real-Time Simulation using the OpenModelica Compiler. In: Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany.
- [7] Bosch Rexroth AG - Motion Logic Programming Interface (MLPI) Documentation
- [8] GNU GCC Release Information, URL: <http://gcc.gnu.org/gcc-3.4/>

# Modular Multi-Rate and Multi-Method Real-Time Simulation

Bernhard Thiele<sup>†</sup>

Martin Otter<sup>†</sup>

Sven Erik Mattsson<sup>‡</sup>

Bernhard.Thiele@dlr.de

Martin.Otter@dlr.de

SvenErik.Mattsson@3ds.com

<sup>†</sup> German Aerospace Center (DLR), Institute for System Dynamics and Control,  
82234 Wessling, Germany

<sup>‡</sup> Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

## Abstract

The demand to ever increase realism and scope of models routinely exceeds the currently available computing power and thus requires thoughts on improving simulation efficiency. This is especially true for real-time simulations, where fixed timing constraints do not allow to just “wait a bit longer”.

This paper presents a new approach in Modelica that allows a modeler to separate a model into different partitions for which individual solvers can be assigned. In effect, this allows to use multi-rate and multi-method time integration schemes that can contribute to improve the efficiency of a (real-time) simulation.

The first part of the paper discusses basic consideration relating to modular (real-)time integration. Afterwards, the implementation of a convenient Modelica library for the partitioning of physical models is briefly described. Finally, the presented library is used to partition a detailed six degree of freedom robot model for modular simulation. The simulation performance of that partitioned model is compared to the simulation performance achieved by using “conventional” global solvers.

*Keywords: multi-rate / multi-method time integration; simulation; clocked discretized continuous-time partitions.*

## 1 Introduction

Testing the actual embedded systems hardware in processor-in-the-loop (PIL) or hardware-in-the-loop (HIL) setups, usually requires that the “virtual” parts of the overall systems are simulated under real-time constraints. This means that the simulation must run as a hard real-time application that always meets its timing deadlines.

Inputs and outputs of a real-time simulation need to be processed at regular intervals. The length of this

interval is called the *simulation frame time*. The *worst case* computation time needs to be less than the simulation frame time. Explicit fixed-step solvers are appropriate numerical integration routines for real-time simulations. Variable-step solvers are generally not appropriate for two reasons: a) because real-time simulation must normally perform I/O operations at regular intervals, and b) because the flexible number of performed simulation model evaluations impede deterministic prediction of worst case computations times.

Implicit fixed-step solvers are problematic in the context of real-time simulation, because the number of iterations required by implicit methods is theoretically unbounded. However, their numerical properties with regard to integrating *stiff systems*<sup>1</sup> is much more favorable compared to explicit solver methods. When integrating stiff systems using explicit methods, the largest possible step size is severely limited due to stability problems of the integration algorithms. Implicit methods can perform much better in such cases. Because of that, attempts have been made to use implicit methods also in real-time simulation. Elmqvist et al. [5, 4] describe techniques that allow to minimize the number of iteration variables for implicit methods (in some cases the number of iteration variables can be reduced to zero!). These algorithms are available in the commercial tool Dymola for real-time simulation purposes under the umbrella term *inline integration* algorithms. The development of linearly implicit or semi-implicit methods are another noticeable attempt in which implicit solver methods are “approximated” by methods that exhibit a bounded number of worst-case iterations. This improves the suitability of these methods for real-time simulation purposes

Computational resources are finite. As a consequence the computational requirements of the real-time simulation must accommodate with available

<sup>1</sup>Stiff systems typically contain dynamically fast and highly damped components.

hardware resources. If the computational load required for the simulation is too high, the simulation model needs to be adapted in order to meet the timing deadlines. Typical options for improving the computational performance include model simplification, use of more efficient algorithms and operations, or replacing computational intensive subsystems with fast table lookups. Another option is to split the simulation model into subsystems, which can be executed in parallel across multiple processors, or which can be executed with different frame rates. The later option is termed *multiframing* or *multi-rate integration* and can be attractive if some subsystems have significantly longer time constants than others. Details about typical techniques used in the context of real-time simulation can be found in relevant literature, e.g., [7, 3].

The new synchronous language elements extension to Modelica [8, Chapter 16] also provides language primitives that allow the developer to partition models into several parts that can be solved separately by different numerical solver methods. A crucial aspect of the partitioning task is to establish adequate *coupling* mechanism between the separate partitions. Following [6] the term *modular simulation* is used to underline that modular coupling approach.

During the partitioning the developer can take advantage of a-priori system knowledge to improve the performance of the simulation. Partitions can be executed with different frame rates and/or can utilize different numerical integration algorithms. In that way *multi-rate* and *multi-method* integration schemes can be realized. To the knowledge of the authors, this is a rather unique feature in a modeling language for physical systems. However, the suitable preparation of simulation models for multi-rate and multi-method integration is still a non trivial task.

In order to facilitate the preparation of simulation models for multi-rate and multi-method integration schemes a Modelica library named MULTIRATE has been build that wraps necessary methodological and technical knowledge in an easy-to-use framework. The theory behind this library, as well as its technical implementation and application to practical problems will be demonstrated in the following sections.

## 2 Clocked Discretized Continuous-Time

Using the synchronous language elements extension [8, Chapter 16] it is possible to define clocks that associate a continuous-time solver to the equations

associated to that clock. This is illustrated in Figure 1. Line 4 defines a periodic clock with the in-

```

model ClockedDiscretizedContinuousTime 1
  Real x(start=0), u; 2
  // 500 ms, no solver: 3
  Clock clk = Clock(5,10); 4
  // 500 ms, ExplicitEuler solver: 5
  Clock clk_solver= 6
    Clock(clk, "ExplicitEuler"); 7
equation 8
  // associate clock clk_solver to u: 9
  u = sample(1.0, clk_solver); 10
  der(x) = -x + u; 11
end ClockedDiscretizedContinuousTime; 12

```

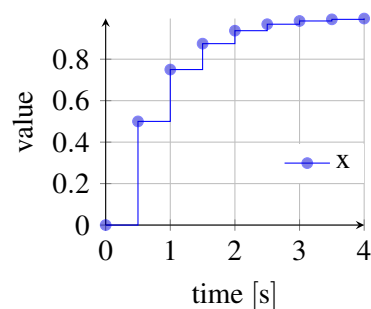


Figure 1: Clocked discretized continuous-time example.

terval 500 ms. Line 7 defines a *solver clock* based on the previously defined clock and assigns as solver method "ExplicitEuler". Other methods that are predefined by the language standard are discussed below. Line 10 associates clock `clk_solver` with the variable `u` (by "sampling" the literal constant 1.0 using clock `clk_solver`). Due to clock inference the differential equation in line 11 can be deduced to be also associated with `clk_solver` and therefore the differential equation needs to be solved by the "ExplicitEuler" method with a fixed-step integration step size of 500 ms. The corresponding plot of `x` is shown at the right-hand side of Figure 1.

The specification [8, Section 16.8.2] defines the conceptual solution algorithms of the predefined methods (tools may provide support for additional solver methods). Since the following discussion is based on these algorithms the respective part from the specifications is reproduced below in a slightly adapted form<sup>2</sup>.

The solvers are defined with respect to the underlying ordinary differential equation in state space form to which the

<sup>2</sup>Most notably, in contrast to the specification text the solver methods are defined in terms of integrating from clock tick  $t_i$  to  $t_{i+1}$ , instead of from  $t_{i-1}$  to  $t_i$ . This is just a simple index shift. The advantage is, that it allows to present some equations in a slightly more concise and readable form.

Table 1: Predefined solver methods for solver clocks

<i>Solver Method</i>	<i>Solution method</i> (for all methods: $y_i := g(x_i, u_i, t_i)$ )
Explicit-Euler	$x_{i+1} := x_i + h \cdot \dot{x}_i$ $\dot{x}_i := f(x_i, u_i, t_i)$
ExplicitMid-Point2	$x_{i+1} := x_i + h \cdot f(x_i + \frac{1}{2}h \cdot \dot{x}_i, \frac{u_i + u_{i+1}}{2}, t_i + \frac{1}{2}h)$ $\dot{x}_i := f(x_i, u_i, t_i)$
Explicit-Runge-Kutta4	$k_1 := h \cdot \dot{x}_i$ $k_2 := h \cdot f(x_i + \frac{1}{2}k_1, \frac{u_i + u_{i+1}}{2}, t_i + \frac{1}{2}h)$ $k_3 := h \cdot f(x_i + \frac{1}{2}k_2, \frac{u_i + u_{i+1}}{2}, t_i + \frac{1}{2}h)$ $k_4 := h \cdot f(x_i + k_3, u_{i+1}, t_{i+1})$ $x_{i+1} := x_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ $\dot{x}_i := f(x_i, u_i, t_i)$
Implicit-Euler	$x_{i+1} := x_i + h \cdot \dot{x}_{i+1}^\dagger$ $\dot{x}_i := f(x_i, u_i, t_i)$
Implicit-Trapezoid	$x_{i+1} := x_i + \frac{1}{2}h \cdot (\dot{x}_i + \dot{x}_{i+1})^\dagger$ $\dot{x}_i := f(x_i, u_i, t_i)$

<sup>†</sup> Equation system with unknowns:  $x_{i+1}, \dot{x}_{i+1}$ .

continuous-time partition can be transformed, at least conceptually ( $t$  is time,  $u(t)$  is the real vector of input variables to the partition,  $x(t)$  is the real vector of continuous-time states, and  $y(t)$  is the real vector of algebraic and/or output variables to other partitions):

$$\begin{aligned}\dot{x} &= f(x, u, t) \\ y &= g(x, u, t)\end{aligned}$$

The solver methods (with exception of "External"<sup>3</sup>) are defined by integrating from clock tick  $t_i$  to clock tick  $t_{i+1}$  and computing the desired variables at  $t_{i+1}$ , with  $h = t_{i+1} - t_i = \text{interval}(u_{i+1})$  and  $x_{i+1} = x(t_{i+1})$ .

Table 1 shows the definitions of the predefined solver methods using the notation from above.

### 3 Multi-Rate

Multi-rate integration can be attractive if some sub-systems have significantly longer time constants than others. This is often the case for multi-domain physical systems since the components of different physical domains often exhibit significant different time constants. A typical example are systems with slow mechanical parts which are controlled by fast electrical

<sup>3</sup>The solver method "External" means that the solution method is defined in the simulation environment and not in the Modelica model.

circuits. If an explicit integration method is used, the numerical stability of the whole system depends on the fastest time constant and it is necessary to choose a respective small step size for integration.

The MULTIRATE library doesn't impose limits on the number of partitions with different frame rates that may be executed together. However, for clarity the basic idea of multi-rate integration is demonstrated with two ODE partitions<sup>4</sup> that will be discretized for two different execution rates:

$$\dot{x}_f(t) = f_f(x_f, x_s, t) \quad (1a)$$

$$\dot{x}_s(t) = f_s(x_f, x_s, t) \quad (1b)$$

The sub-index  $f$  stands for the "fast" partition and  $s$  stands for the "slow" partition. Using the ExplicitEuler method from Table 1 for discretization results in a system of recurrence equations of the form:

$$\begin{aligned}x_f(t_i + j \cdot h_f) &= x_f(t_i + (j-1) \cdot h_f) \\ &+ h_f \cdot f_f(x_f(t_i + (j-1) \cdot h_f), \\ &x_s(t_i + (j-1) \cdot h), t_i + (j-1) \cdot h) \quad (2a) \\ x_s(t_{i+1}) &= x_s(t_i) + k \cdot h_f \cdot f_s(x_f(t_i), x_s(t_i), t_i) \quad (2b)\end{aligned}$$

where  $k$  and  $j$  are integers,  $k$  is the ratio of the two step sizes,  $j = 1 \dots k$ ,  $h_f$  is the step-size of the fast partition, and  $t_{i+1} - t_i = k \cdot h_f = h_s$  is the step-size of the slow partition.

Note that Equation (2a) does not specify how  $x_s(t_i + (j-1) \cdot h)$  is calculated. Equation (2b) is not defined for intermediate values between  $t_i$  and  $t_{i+1}$ . Therefore, an interpolation or extrapolation scheme needs to be used to estimate the intermediate values.

For real-time simulation the sequence in which the computation of fast and slow frames are interspersed is important. Ledin [7] describes three typical execution schemes. The timing diagram in Figure 2 shows that execution schemes by means of an example where the ratio of the slow and the fast step size,  $k$ , is  $k = 3$ . The three schemes are described briefly below:

1. *Multiframing in a single task with no fast-frame real-time I/O.* The slow frame rate is treated as a "master" frame rate in which the slow frame is executed first, followed by a burst of the  $k$  fast frames. This scheme is only acceptable if the fast frames do not perform any real-time I/O.

<sup>4</sup>In general, partitions in the MULTIRATE library may consist of differential and algebraic equations and the partitions are coupled over designated input and output variables, but this is omitted here in favor of a more succinct presentation of the basic idea of multi-rate integration.

2. *Multiframing in a single task with fast-frame real-time I/O.* The fast frames are executed at fixed intervals of  $h_f$  length. The computations needed in the slow frame are split into several subframes which are interspersed after the fast frame calculations. However, splitting the slow frame into several suitable subframes is rarely a simple thing to do. This is a serious drawback of this method.
3. *Multiframing in a multitasking environment with rate monotonic scheduling (RMS).* In this case the scheduler will give CPU access to the task with the higher priority (computation of fast frames) and interrupt the lower priority task (computation of slow frames) until the higher priority task has finished its computations. During the times in which the higher priority task is idle, the CPU access is given back to the lower priority task to resume its computations. No (manual) splitting into subframes is needed which is a huge advantage compared to the previous method.

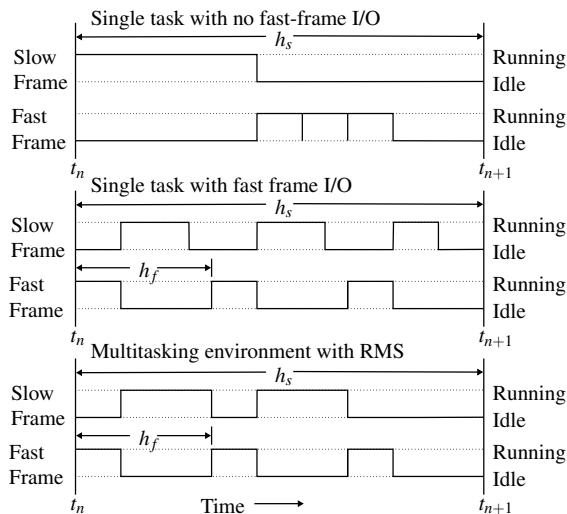


Figure 2: Three different multiframing schemes for real-time simulation.

## 4 Multi-Method

*Multi-method integration* (also called *mixed-mode integration*) is yet another option to improve the execution performance of (real-time) simulation. In contrast to multi-rate integration that uses different integration step sizes for distinct partitions, multi-method integration uses different integration methods for distinct partitions. Similarly to multi-rate integration, multi-

method integration can be attractive if some partitions have significantly longer time constants than others.

A typical scenario is to split a system into fast parts and slow parts and use an implicit integration method for the fast parts and an explicit integration method for the slow parts. Schiela and Olsson [11] describe such a mixed-mode integration scheme (using explicit and implicit Euler methods) in which they employ an automatic partitioning approach based on linearization and eigenvalue analysis. This disburdens the developer from partitioning the system. However, if a system is highly nonlinear, inspecting eigenvalues becomes questionable since the eigenvalues of the linearized system move around with time. A user controlled partitioning, leveraging a-priori system knowledge, can be more adequate and effective in such cases.

Similar to multi-rate integration, the basic idea of multi-method integration is demonstrated on the basis of two ODE partitions in the form of equation system (1). Using the `ImplicitEuler` method from Table 1 for the “fast” partition and the `ExplicitEuler` method for the “slow” partition results in recurrence equations of the form:

$$x_s(t_{i+1}) = x_s(t_i) + h \cdot f_s(x_f(t_i), x_s(t_i), t_i) \quad (3a)$$

$$x_f(t_{i+1}) = x_f(t_i) + h \cdot f_f(x_f(t_{i+1}), x_s(t_{i+1}), t_{i+1}) \quad (3b)$$

where  $h = t_{i+1} - t_i$  is the integration step-size. At first,  $x_s(t_{i+1})$  is computed using the explicit Euler method. This value is afterwards used to compute  $x_f(t_{i+1})$  using the implicit Euler method.

The `MULTIRATE` library allows to combine any of the solver methods listed in Table 1. Note that it is easily possible to combine multi-rate and multi-method integration within the framework of the `MULTIRATE` library. This allows to exceed the benefits compared to applying the methods separately.

## 5 Partition Coupling

An important aspect when applying multi-rate and multi-method integration is the coupling between the involved partitions. In the context of the framework provided by the `MULTIRATE` library the coupling scheme of Figure 3 depicts the basic idea (liberally abstracting from the details). Note that although the coupling is discussed by considering the special case of two coupled partitions, the basic principles carry over to the general case of  $n$  partitions.

Partition 1 and 2 may be discretized by using one of the solver methods defined in Table 1. Different solver



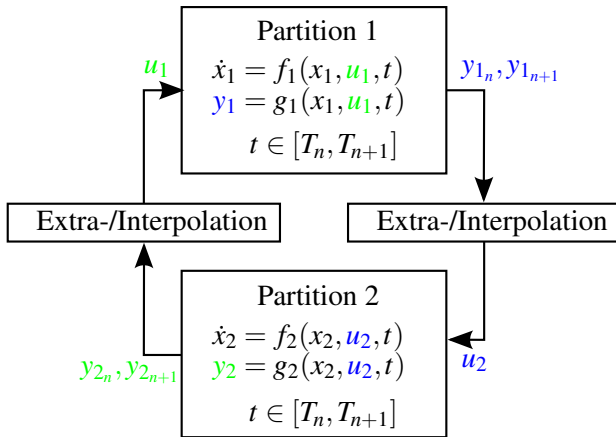


Figure 3: Coupling of two partitions. Communication takes place at discrete time instants  $t = T_0, T_1, \dots, T_k$ . Depending on the applied coupling scheme the approximation of the coupling terms  $u_1, u_2$  can be based on interpolation (if  $y_{1_{n+1}}$ , or respectively  $y_{2_{n+1}}$  is available) or it must be based on extrapolation.

methods, as well as different step sizes can be used as long as the ratio of the step sizes is an integer.

Data exchange between clocked discretized continuous-time partitions is only possible at clock ticks. Therefore, the clock ticks of the slower partition determine the discrete time grid in which data is exchanged between the two partitions. This is quite similar to the situation encountered in co-simulation scenarios, in which the communication and data exchange between two distinct systems is restricted to discrete synchronization points  $T_n$ .

However, in contrast to co-simulation the modular integration considered in the MULTIRATE framework has some distinctive characteristics:

- It inherits the characteristics of the synchronous model of computation that has been introduced in the “Synchronous Language Elements” extension in Modelica 3.3 [8, Chapter 16]. This has the advantage that the formal model of various coupling schemes can be expressed in a high-level, declarative manner which is close to the underlying conceptual mathematical model. However, the drawback is that optimizations that require a more low-level control can not be realized.
- Modelica uses *acausal* connectors to assemble models of physical components. However, the coupling of partitions according to Figure 3 requires *causal* data-flow. It is not obvious how convenient and effective coupling schemes can be

realized when a model should be partitioned at acausal connectors.

## 5.1 Mathematical Model

For the further analysis more detailed mathematical models than the one indicated in Figure 3 are proposed. As before, the discussion is based on two partitions, but carries over to more general settings involving  $n$  partitions.

Figure 4 depicts a continuous-time domain model for two coupled partitions, including inputs  $(u_1, u_2)$  and outputs  $(y_1, y_2)$  due to real-time I/O hardware devices. The dynamics of the coupled partitions are modeled as differential-algebraic equations (DAEs) in autonomous semi-explicit form

$$\dot{x}_1 = f_1(x_1(t), \tilde{x}_2(t), z_1(t), \tilde{z}_2(t), \tilde{u}_1(t)) \quad (4a)$$

$$0 = \gamma_1(x_1(t), \tilde{x}_2(t), z_1(t), \tilde{z}_2(t), \tilde{u}_1(t)) \quad (4b)$$

$$y_1 = g_1(x_1(t), \tilde{x}_2(t), z_1(t), \tilde{z}_2(t), \tilde{u}_1(t)) \quad (4c)$$

$$\dot{x}_2 = f_2(\tilde{x}_1(t), x_2(t), \tilde{z}_1(t), z_2(t), \tilde{u}_2(t)) \quad (4d)$$

$$0 = \gamma_2(\tilde{x}_1(t), x_2(t), \tilde{z}_1(t), z_2(t), \tilde{u}_2(t)) \quad (4e)$$

$$y_2 = g_2(\tilde{x}_1(t), x_2(t), \tilde{z}_1(t), z_2(t), \tilde{u}_2(t)) \quad (4f)$$

with differential variables  $x_i \in \mathbb{R}^{n_{x_i}}$ , algebraic variables  $z_i \in \mathbb{R}^{n_{z_i}}$ , (real-time) inputs  $u_i \in \mathbb{R}^{n_{u_i}}$ , and outputs  $y_i \in \mathbb{R}^{n_{y_i}}$  where  $n_{x_i}, n_{z_i}, n_{u_i}, n_{y_i} \in \mathbb{N}$  and consistent initial conditions  $x_i(t_0) = x_{i,0}, z_i(t_0) = z_{i,0}$ . The (continuous-time) variables with tilde,  $\tilde{x}_i, \tilde{z}_i, \tilde{u}_i$  are reconstructed from a number of  $m \geq 1$  sampled (discrete-time) values of the variables without tilde of the same name by means of extrapolation or interpolation using a “reconstruction” operator denoted  $\Psi$

$$\tilde{x}_i(t), \tilde{z}_i(t) = \Psi_i(\chi_i, \zeta_i)(t) \quad t \in [T_n, T_{n+1}] \quad (4g)$$

$$\tilde{u}_i(t) = \Psi_{u_i}(v_i)(t) \quad (4h)$$

where  $\chi_i, \zeta_i, v_i$  are sampled at time instants  $t_k \in (T_{n-m}, T_n]$  (extrapolation), or  $t_k \in (T_{n+1-m}, T_{n+1}]$  (interpolation)

$$\chi_{i_k}, \zeta_{i_k} = x_i(T_k), z_i(T_k) \quad k = 1 \dots k_i, \quad T_k < T_{k+1},$$

$$k_i \in \mathbb{N}, \quad \chi_i \in \mathbb{R}^{n_{x_i} \times k_i}, \quad \zeta_i \in \mathbb{R}^{n_{z_i} \times k_i}$$

$$v_{i_k} = u_i(T_k) \quad k = 1 \dots k_{u_i}, \quad T_k < T_{k+1},$$

$$k_{u_i} \in \mathbb{N}, \quad v_i \in \mathbb{R}^{n_{u_i} \times k_{u_i}}.$$

The operator  $\Psi$  is loosely borrowed from the mathematical framework described in [12, p. 1495], where it is defined as extrapolation operator. The definition there is mathematically more technical and rigorous than considered necessary for this work.

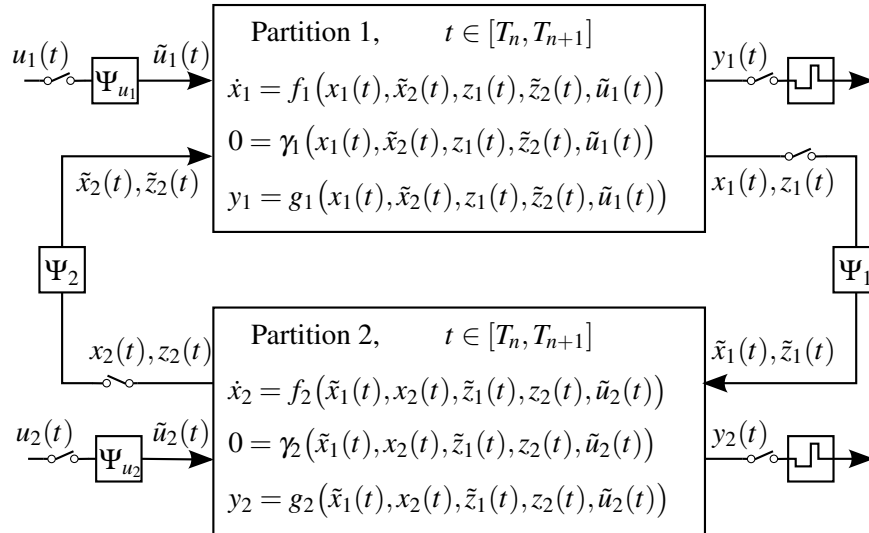


Figure 4: Illustration of the used continuous-time domain mathematical model for partition coupling, including external real-time I/O ( $u_1, u_2, y_1, y_2$ ). The inputs to the partitions are first sampled at discrete time instants  $t = T_0, T_1, \dots, T_k$ , subsequently the operators  $\Psi_i$  are applied to the (time-discrete) signals in order to provide extrapolated/interpolated continuous-time signals during a period of continuous-time system evolution ( $t \in [T_n, T_{n+1}]$ ).

Data exchange between partitions and the update of coupling terms is restricted to the time-discrete synchronization points  $t = T_0, T_1, \dots, T_k$ . In co-simulation the steps from  $T_n \rightarrow T_{n+1}$  are referred to as *macro steps*. The union of all macro-time steps is a macro-time grid where the partitions update their coupling terms. Generally, the real-time inputs and outputs of the respective partitions may be sampled at discrete time instants that are different to the macro-time grid instants. During a macro step  $T_n \rightarrow T_{n+1}$  the dynamics of the partitions evolve according to the governing DAE using extrapolated (or interpolated) data. Note that  $\tilde{x}_i, \tilde{z}_i, \tilde{u}_i$  are continuous in each macro step  $T_n \rightarrow T_{n+1}$  but may have jump discontinuities at the synchronization points  $T_n$ .

Since this work is concerned with real-time simulation, it is natural to consider only equidistant macro-time grids with constant macro-step size  $h = (T_{n+1} - T_n)$ . Moreover, the utilized synchronous framework in conjunction with the discretization formula given in Table 1 suggests to describe the coupling within a (discrete-time) recurrence equation framework. The step-sizes  $h_i$  of the two partitions may differ, but the ratio between the slower and faster period must be an integer multiple. Without loss of generality assume that  $h_1$  is the faster partition and denote  $N = h_2/h_1$  as the frame ratio. The partitions are synchronized at the discretization points ( $\equiv$  macro-time grid)

$$t = k \cdot N \cdot h_1 = k \cdot h_2 = k \cdot h, k \in \mathbb{N}. \quad (5)$$

At these points the equations of both partitions have to be fulfilled concurrently (synchronous model of computation).

The overall discretized system equations can now be described in terms of the faster sampling period  $h = h_1$ , further also denoted as micro-time step. The precise time dependencies, i.e., at which instant  $t_i = i \cdot h_1$  on the micro-time grid coupling variables from time instant  $t_{i < j}$  are required depend on the utilized discretization method. Table 2 shows the dependencies for the solver methods supported by the Modelica standard. The important characteristics to be observed are:

1. Algebraic equations always, regardless of the utilized solver method, require the variable values at the current time instant on the micro/macro-time grid.
2. For state integration using `ExplicitEuler` coupling variable values and state variable values from previous activation times are sufficient.
3. The other two explicit methods require coupling variable values from the *current* time instant (due to the occurrence of term  $\frac{u_i + u_{i+1}}{2}$  in the defining equations in Table 1!)
4. The implicit methods require coupling variable values and state variable values from the current time instant.

Table 2: Coupled variables time dependencies after discretization

Solver Method	Time instant dependencies
	For all methods: $h_1$ is the step-size of the fast partition, and $t_{i+1} - t_i = N \cdot h_1 = h_2 = h$ is the step-size of the slow partition, $N, j \in \mathbb{N}, N = h_2/h_1, j = 1 \dots N$ . $\tilde{x}_{1,i}, \tilde{z}_{1,i}, \tilde{x}_{2,i}, \tilde{z}_{2,i}$ are approximated from $x_{1,i}, z_{1,i}, x_{2,i}, z_{2,i}$ by a suitable extrapolation method. Time dependencies in the algebraic equations are always: $0 = \gamma_1(x_{1,iN+j}, \tilde{x}_{2,iN+j}, z_{1,iN+j}, \tilde{z}_{2,iN+j})$ $0 = \gamma_2(\tilde{x}_{1,(i+1)N}, x_{2,(i+1)N}, \tilde{z}_{1,(i+1)N}, z_{2,(i+1)N})$
Explicit-Euler	$x_{1,iN+j} = f_1(x_{1,iN+j-1}, \tilde{x}_{2,iN+j-1}, z_{1,iN+j-1}, \tilde{z}_{2,iN+j-1})$ $x_{2,(i+1)N} = f_2(\tilde{x}_{1,iN}, x_{2,iN}, \tilde{z}_{1,iN}, z_{2,iN})$
Explicit-MidPoint2 / RungeKutta4	$x_{1,iN+j} = f_1(x_{1,iN+j-1}, \tilde{x}_{2,iN+j}, z_{1,iN+j-1}, \tilde{z}_{2,iN+j})$ $x_{2,(i+1)N} = f_2(\tilde{x}_{1,(i+1)N}, x_{2,iN}, \tilde{z}_{1,(i+1)N}, z_{2,iN})$
Implicit-Euler / Trapezoid	$x_{1,iN+j} = f_1(x_{1,iN+j}, \tilde{x}_{2,iN+j}, z_{1,iN+j}, \tilde{z}_{2,iN+j})$ $x_{2,(i+1)N} = f_2(\tilde{x}_{1,(i+1)N}, x_{2,(i+1)N}, \tilde{z}_{1,(i+1)N}, z_{2,(i+1)N})$

This has the following consequences for clocked discretized continuous-time partitions that are coupled within Modelica’s synchronous computation framework:

- In the general case, there is no scheduling of  $\gamma_1$  and  $\gamma_2$ , that satisfies reciprocal data dependencies without resorting to extrapolation from previous values.
- State integration using `ExplicitEuler` allows to use  $\tilde{x}_{1,iN}, \tilde{z}_{1,iN} = x_{1,iN}, z_{1,iN}$  (no extrapolation needed, since values already available at  $t_{i+1}N$ ). However,  $\tilde{x}_{2,iN+j-1}, \tilde{z}_{2,iN+j-1}$  need extrapolation for  $j > 1$ . Scheduling of  $f_1, f_2$  at the macro-time grid is always possible, since only past values are required at these points.
- For state integration using the remaining explicit and implicit methods, there is no scheduling of  $f_1$  and  $f_2$  that satisfies reciprocal data dependencies in the general case without resorting to extrapolation from previous values.

Note that in Modelica’s synchronous computation framework it is not allowed to have algebraic loops *spanning* clocked discretized continuous-time partitions (however, it is allowed to have algebraic loops *within* a partition!). Therefore, there must be a sorting for the coupling equations at macro-time grid points that allows to evaluate them in a sequential order that satisfies data dependencies.

The previous discussion already allows to identify some of the consequences when using the synchronous framework for partition coupling:

- A staggered method (Gauss-Seidel scheme) that would first integrate the slow partition, extrapolating the inputs from the coupled (fast) partition, and after that integrate the fast partition, *interpolating* from the results of the slow partition, is not feasible. This is because within the synchronous framework new values are accessible only at the points at time where they are *valid* and not directly after they have been computed. Also, it is not possible for the modeler to directly influence the sequence of calculations. This is at the discretion of the simulation tool that will only guarantee to respect data flow dependencies.
- Nevertheless, an execution scheme similar to the “*Single task with no fast-frame real-time I/O*” depicted in Figure 2 is feasible, however, the interpolation of coupling variables during execution of the fast partitions is not possible (extrapolation is required).
- The synchronous model of computation makes the abstractions that equations at time instants are evaluated instantaneously. However, in reality, computation takes time. From a real-time I/O timing perspective it is desirable that the simulation time instants of inputs and outputs closely fit the real-time instants. For that reason, dedicated real-time integrators are typically designed to require only past inputs for the integration up to the current time instant. This allows to compute the output values required at real-time instants  $t = T_n$  during a computational period scheduled at  $t < T_n$ . This computational timing details are beyond the scope of the Modelica specification. However, observe that all solvers specified in Table 1 (except `ExplicitEuler`) require the (real-time) inputs at  $t = T_n$  in order to compute the outputs at  $t = T_n$ . Therefore, the real-time outputs will be inevitable afflicted with a (potentially considerable) computational delay ( $t_{\text{outputs}} = T_n + t_{\text{delay}}$ ).

Obviously, it will often be desirable to keep at least  $t_{\text{delay}}$  as small as possible. For the “*Single task with no fast-frame real-time I/O*” scenario it is therefore advisable to schedule the computation of the slow frame’s real-time outputs before executing the fast frames (just as depicted in Figure 2)<sup>5</sup>.

- Parallel coupling methods using solely extrapolated coupling terms (Jacobi-scheme) can be realized within the available synchronous framework. Therefore, parallel execution of frames as depicted at the bottom of Figure 2 should be feasible. This is expected to be particularly attractive if a simulation can benefit from multi-core hardware and distribute the computational load on the available cores<sup>6</sup>.
- Some of the solver methods in Table 1 are *multi-pass* methods<sup>7</sup>, i.e., they require several “intermediate” evaluation of  $f(\cdot)$  during integration. Since communication within the synchronous framework can only occur at clock ticks, the needed intermediate values of the inputs  $u$  are computed by interpolation. The alternative to actually acquire (real-time) samples of  $u$  for this intermediate evaluations is not possible in the current framework. However, it should be noted that for dedicated real-time integrators the use of intermediate input samples is typically considered [1, 9].

## 5.2 Partition Coupling at Acausal Connectors

In Figures 3 and 4 the coupling of partitions is accomplished by causal (directed) dataflow. However, physical modeling in Modelica relies on acausal connectors. It is not obvious how to accomplish partition coupling at the boundary of acausal connectors.

The following discussion is based on couplings at the boundary of rotational mechanical connectors from the Modelica Standard Library. However, the

<sup>5</sup>The Modelica standard doesn’t provide any possibility for a modeler to control the scheduling of computations. Therefore, a reasonable scheduling of real-time I/O can be seen as an implementation quality trait of a Modelica tool.

<sup>6</sup>However, the Modelica standard doesn’t provide any means for a modeler to control whether computations are parallelized. At the time of writing this article the authors are not aware of any Modelica tool that supports parallelization of clocked partitions.

<sup>7</sup>Namely, the solver methods `ExplicitMidPoint2` and `ExplicitRungeKutta4`.

presented principles are easily transferable to other physical connectors.

Consider the academic example model in Figure 5 and assume the model shall be partitioned somewhere between the inertias  $J_1$  and  $J_2$ .

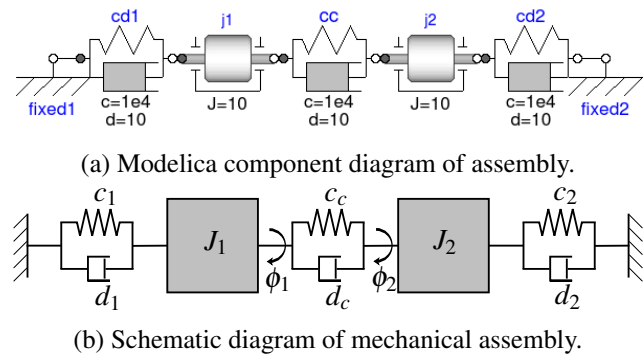


Figure 5: The example model: a linear 2-DOF oscillator.

Figure 6 shows two common approaches to partition the model for modular integration [2]:

**Force/displacement coupling** Partition  $P_1$  provides the cut torque at its boundary as output which is the input to partition  $P_2$ . Conversely, partition  $P_2$  provides the displacement at its boundary as output which in turn is the input to partition  $P_1$  (Figure 6a).

**Displacement/Displacement coupling** The coupling force element of  $S_1$  is duplicated in  $S_2$ . The two partitions are coupled by the displacements of  $S_1$  and  $S_2$  (Figure 6b).

## 5.3 Implementation in Modelica

The Modelica MULTIRATE library provides convenient building blocks for partitioning a model for multi-rate and multi-method simulation. It is implemented on top of the synchronous language elements, partly reusing functionality provided by the MODELICA\_SYNCHRONOUS library [10].

### 5.3.1 Force/Displacement Coupling

The component diagram in Figure 7a shows the oscillator from Figure 5a in a *force/displacement* coupling configuration. The coupling component `subSample1` is an instance of the `SubSampleForceDisp` class from the MULTIRATE library. The `SubSampleForceDisp` class provides a few parameters to modify the coupling characteristics, namely `inferFactor` to define that the

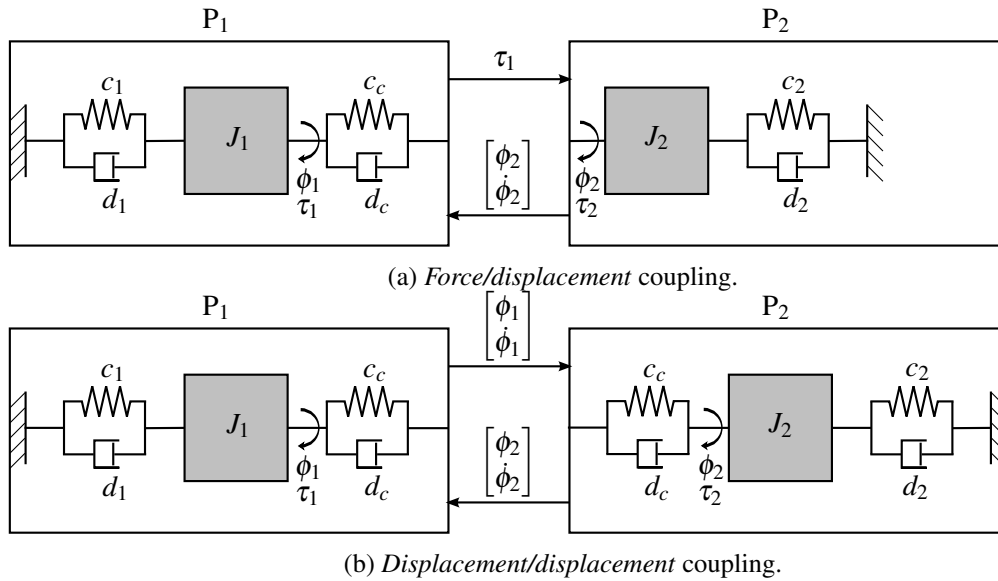


Figure 6: Partitioning the example model.

tool shall determine the sub-sample factor by clock inference. Otherwise, the sub-sample factor can be entered manually. Efficient polynomial extrapolation of arbitrary degree  $nP$  is supported for extrapolating values stemming from the slow partition during execution of the fast partition<sup>8</sup>. With default settings the output of the fast partition will be delayed one clock tick, however parameter `useDirectFeedthrough` allows to avoid that delay. The delay is necessary if both partitions need at macro-time steps  $t = iN$  the current coupling values of the respective other partition. As has been discussed in Section 5.1 this reciprocal data dependencies lead to illegal algebraic loops spanning the coupled partitions. Note that if at least one of the partitions uses `ExplicitEuler` as solver method, a scheduling without algebraic loops may become feasible and `useDirectFeedthrough` may be set to `true`.

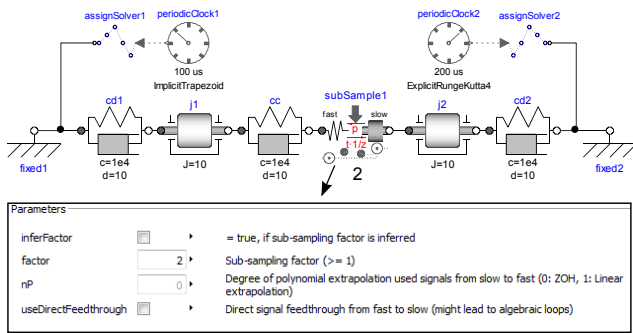
The design of the icon of the `SubSampleForceDisp` class gives a visual clue regarding the intended placement within a model. The component at the left hand side receives a displacement ( $\phi$  and derivatives of  $\phi$ ), and needs to provide a torque  $\tau$ . This is typical for spring like elements. The right hand side component receives a torque and reacts with a displacement. This is typical for inertia like components. The sub-

sampling factor is displayed at the bottom of the icon (provided that `inferFactor=false`).

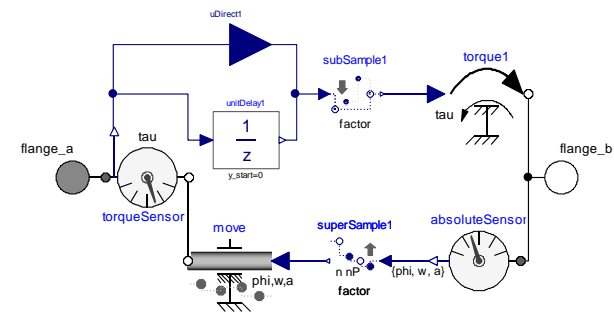
The internal structure of the `SubSampleForceDisp` class is depicted in Figure 7b. The parallel branches with component `uDirect1` and `unitDelay1`<sup>9</sup> are conditional branches. Their activation is mutually exclusive and depends on the value of the parameter `useDirectFeedthrough`. The `subSample1` block wraps Modelica's `subSample(..)` operator, which performs fast-to-slow rate transitions. Component `absoluteSensor` returns an array with the displacement variables  $\{\phi, \dot{\phi}, \ddot{\phi}\}$ . Upsampling and (polynomial) extrapolation of the variables is performed by the `superSample1` block. Since  $\phi$  is a discrete-time (sampled) signal (see Figure 4), it carries no information about its derivatives. It is the task of component `move` to force the movement of flange `flange_a` according to signals  $\phi$ ,  $\dot{\phi}$  and  $\ddot{\phi}$ . In the implementation smoothness information of  $\phi$  is recovered at the sampling points by using the sampled values of  $\dot{\phi}$  and  $\ddot{\phi}$  and setting the recovered signal equal to `flange_a.phi`. This is accomplished by using the derivative annotation as described in [8, Section 17.7, "Declaring Derivatives of Functions"]. Components `torqueSensor` and `torque1` are from the Modelica Standard Library.

<sup>8</sup>Note that  $nP=0$  is equivalent to holding the value constant and  $nP=1$  is equivalent to linear extrapolation. Moreover, while increasing the order of extrapolation may result in improved numerical stability and accuracy in some applications it may also deteriorate numerical stability (see [2] for comprehensive numerical experiments regarding extrapolation and interpolation of coupling signals in co-simulation scenarios). In many cases,  $nP=0$  or  $nP=1$  seems to be a good choice.

<sup>9</sup>The `unitDelay1` block wraps the `previous` operator. According to the current specification [8, Section 16.8.1] the use of `previous` within a clocked discretized continuous-time partition is forbidden. However, the Dymola 2014 FD01 tool used for this work is more lenient and allows it. This is a desirable feature in order to enable the described implementation.



(a) Example model using a *force/displacement* coupling in a multi-rate and multi-method configuration. The coupling component subSample1 is an instance of class SubSampleForceDisp from the MULTIRATE library.



(b) Component composition diagram of class SubSampleForceDisp. This class implements the *force/displacement* coupling.

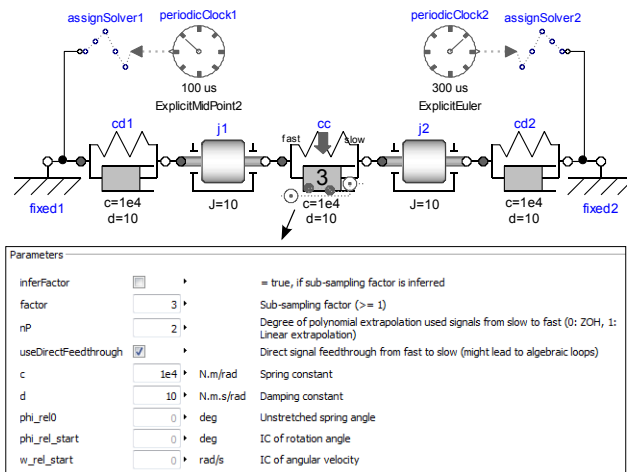
Figure 7: *Force/displacement* coupling using the MULTIRATE library.

### 5.3.2 Displacement/Displacement Coupling

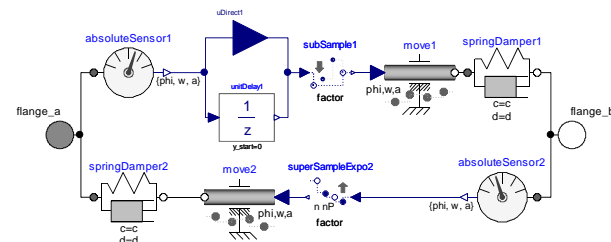
The component diagram in Figure 8a shows the oscillator in a *displacement/displacement* coupling configuration. The component cc is an instance of class SubSampleDispDisp from the MULTIRATE library. It constitutes both: the dynamic equations of motion for the spring/damper element and the partition coupling equations.

The upper part of parameters of the SubSampleDispDisp class is identical to the parameters provided by the SubSampleForceDisp class. The lower part provides parametrization for the spring/damper element.

The internal structure of the SubSampleForceDisp class is depicted in Figure 8b. Note that the dynamic equations for the spring/damper element are duplicated: while component springDamper1 is assigned to the slow partition, springDamper2 is assigned to the fast partition. This “overlapping” integration often leads to more favorable numerical stability properties (see Busch [2]). Aside from this, the components appearing in Figure 8b are already known from Fig-



(a) Example model using *displacement/displacement* coupling in a multi-rate and multi-method configuration.



(b) Implementation of *displacement/displacement* coupling class.

Figure 8: *Displacement/displacement* coupling using the MULTIRATE-library.

ure 7b.

While this section described the partitioning at the example of 1-dimensional, rotational mechanics, it is needless to say that the basic approach carries over to other physical domains. Furthermore, the mindful reader may miss a SubSampleDispForce class. That class was omitted, since it basically results by swapping and adapting respective components in Figure 7b.

## 6 Application to a 6-DOF Robot Model

In order to understand whether the partitioning is suitable for “real-world” systems with considerable complexity, the RobotR3 example (a detailed model of a robot with six degrees of freedom) from the MultiBody package of the Modelica Standard Library (MSL) was adapted and partitioned into three parts (see Figure 9):

1. “clockControl” partition. The partition consists of a clocked discrete-time path planning component and clocked discrete-time P-PI cas-

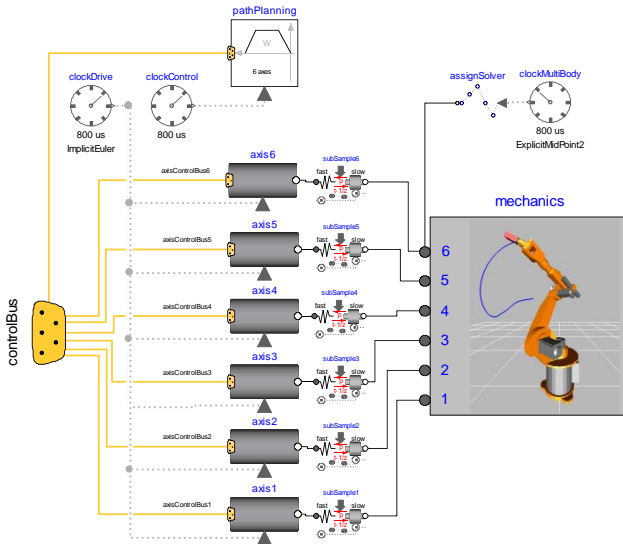


Figure 9: 6-DOF robot example adapted from the RobotR3 example of the multi-body package of the MSL.

cade controllers for the six axes (inner PI-controllers to control the motor speeds, and outer P-controllers to control the motor positions). Therefore, the continuous-time controllers and the path planning component from the original RobotR3 example were replaced by a discrete-time (digital) implementation. The sample period of that partition is set to  $800 \mu\text{s}$ .

2. **“clockDrive” partition.** The remaining parts in each axis (motor including current controller and the gearbox including gear elasticity and bearing friction) are combined into a clocked discretized continuous-time partition. The steady-state initialization found in the original RobotR3 example was removed since the initialization of clocked partitions differs from the standard scheme of initialization in Modelica [8, Section 16.9, “16.9 Initialization of Clocked Partitions”]. Instead, compatible initial values have been set at appropriate places. “ImplicitEuler” with a step size of  $800 \mu\text{s}$  is used as solver method.
3. **“clockMultiBody” partition.** Except for setting compatible initial values, the multi-body part is identical to the original RobotR3 model. “ExplicitMidPoint2”, also with a step size of  $800 \mu\text{s}$ , is used as solver method.

Partitions “clockDrive” and “clockMultiBody” are coupled at the mechanical flanges connecting the axes

with the mechanics multi-body system by *force/displacement* components with constant extrapolation ( $nP=0$ ) and no direct feedthrough.

The simulation performance using the solver clocks was compared against simulations performed on the same model (using various solvers), but without using coupled clocked discretized continuous-time partitions (see Figure 10).

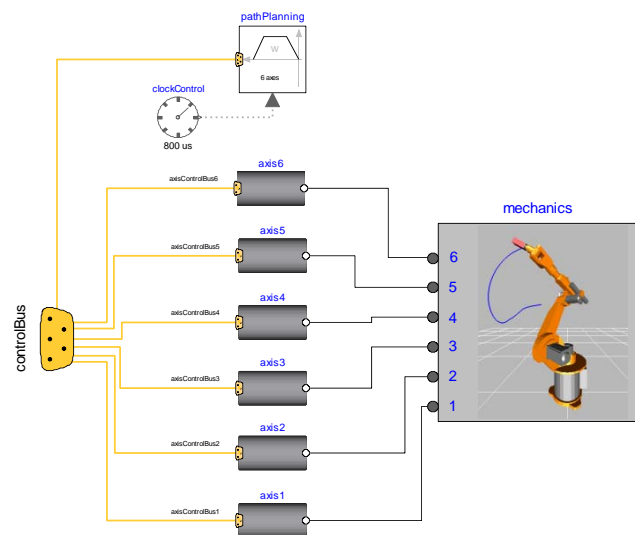


Figure 10: Comparison model: 6-DOF robot example without coupled clocked discretized continuous-time partitions.

The numerical experiment is conducted on a notebook with an Intel Core 2 Duo CPU P9700 @ 2.8 GHz and 4.0 GB of RAM. The simulation tool is Dymola 2014 FD01 running on a 64-bit Microsoft Windows 7 operating system.

The reference simulation result is obtained by simulation of the comparison model (Figure 10) using DASSL as integrator. The solver parameter “Tolerance” is set to 0.0001 for all simulation runs. The simulation interval is always set to  $[0, 2]$  seconds. The step size of the tested solver methods is iteratively increased until either integration fails (which for the considered model typically means that too large residuals appear while solving (nonlinear) systems of equations), or the simulation result deviates considerably from the DASSL reference solution.

The decision whether a result deviates considerable from the reference solution is made by two criteria: a) visual inspection of the trajectory of the load at the robot arm tip, and b) deviation of the trajectory to the

reference solution defined by the norms

$$E_2 = \sqrt{\int_{t_0}^{t_e} e_r(t)^2 dt} \quad (6a)$$

$$E_\infty = \sup_{t \geq t_0} (e_r(t)) \quad (6b)$$

where  $t_0$  is the simulation start time,  $t_e$  is the simulation stop time and

$$e_r(t) = \frac{\|r(t) - r_{\text{DASSL}}(t)\|}{\|r_{\text{DASSL}}(t)\|} \quad r, r_{\text{DASSL}} \in \mathbb{R}^3$$

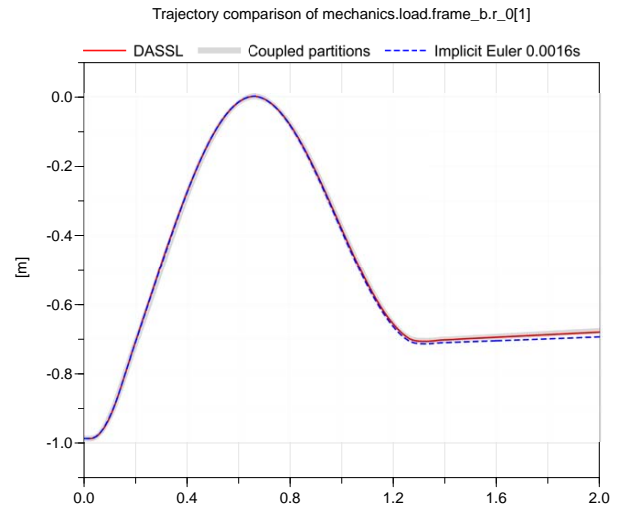
provides a relative error measurement by relating the distance between the obtained solution  $r(t)$  and the reference solution  $r_{\text{DASSL}}(t)$  to the magnitude of the vector  $r_{\text{DASSL}}(t)$ . For the actual computation of  $E_2$  and  $E_\infty$ , (6a) and (6b) are numerically evaluated over an uniformly spaced grid with spacing  $\Delta t = 0.0008$ . Therefore, the integral in (6a) is approximated by numerical summation.

In Figure 11 the DASSL reference solution is compared with the result when simulating the coupled partitions model of Figure 9 and the result when simulating the comparison model of Figure 10 with an inline implicit Euler solver.

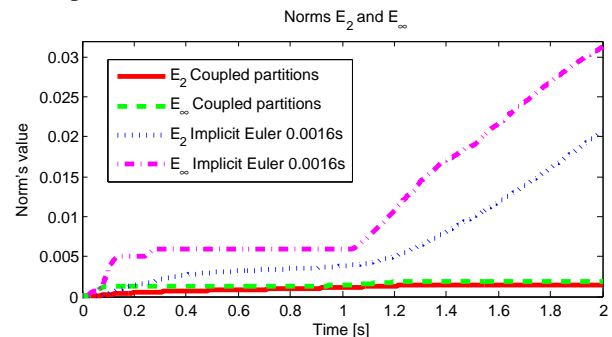
Figure 11a shows the first Cartesian coordinate of the trajectory of the robot arm tip. It can be observed that the solution computed by the implicit Euler solver with fixed step size 0.0016 s diverges considerably at the end of the simulation run. Examining the evolution of the norms  $E_2$  and  $E_\infty$  confirms that observation (see Figure 11b). At  $t = 2$  s the relative error of the implicit Euler solution compared to the DASSL reference is about 3%.

Beside using the DASSL integrator, various real-time inline integrators provided by the tool Dymola [5] were tested with the comparison model. Table 3 summarizes the results obtained by that simulation experiments and contrasts them to the result obtained by simulation of the coupled partitions model of Figure 9.

The *inline implicit Euler* solver displayed the best performance of the tested “conventional” solvers. However, for this scenario it was possible to even outperform that solver by a factor of about 2.9 (at comparable accuracy to the reference solution) by using a coupling of clocked discretized continuous-time partitions in combination with judiciously selected solver methods.



(a) Trajectory of the first Cartesian coordinate of the robot arm tip.



(b) Evolution of the norms  $E_2$  and  $E_\infty$  over the simulation time.

Figure 11: Simulation results of coupled partition approach and implicit Euler solver (step size 0.0016 s) compared to the DASSL reference solution.

## 7 Conclusions

This paper presented a new approach in Modelica that allows a modeler to separate a model into different partitions for which individual solvers can be assigned. This effectively allows multi-rate and multi-method time integration schemes that can improve simulation efficiency in certain cases. Additionally, there is a potential to execute the partitions in parallel to gain further simulation speedups. However, this is not supported by currently available tools.

The approach is based on *clocked discretized continuous-time partitions*, a concept that was introduced as part of the synchronous language elements extension into the Modelica 3.3 language standard. However, until now it has not been applied in the context considered in this article.

The article started with a formal description of



Table 3: Comparison of solver methods

Solver	Step size (s)	CPU-time for integration (s)	$E_2$	$E_\infty$
DASSL		10.8	0	0
<b>Coupled partitions</b>	$2 \times \mathbf{0.0008}$	<b>0.7</b>	0.0014	0.0019
<b>Inline implicit Euler</b>	<b>0.0008</b>	<b>2.0</b>	0.0058	0.0081
<i>Inline implicit Euler (considerable deviation)</i>	<i>0.0016</i>	<i>(1.2)</i>	0.0207	0.0313
Inline trapezoidal	0.0008	2.1	0.0059	0.0082
<i>Inline trapezoidal (considerable deviation)</i>	<i>0.0016</i>	<i>(1.3)</i>	0.0208	0.0313
Inline explicit Euler	0.00002	2.3	0.0009	0.0017
<i>Inline explicit Euler</i>	<i>0.00004</i>	<i>(Failed)</i>		
Mixed explicit/implicit Euler	0.0004	3.5	0.0004	0.0008
<i>Mixed explicit/implicit Euler</i>	<i>0.0008</i>	<i>(Failed)</i>		
Inline explicit RK 4	0.00002	8.6	0.0004	0.0009
<i>Inline explicit RK 4</i>	<i>0.00004</i>	<i>(Failed)</i>		

the mathematical prerequisites of coupling partitions within the synchronous language elements framework with special regard to timing requirements inherent to real-time simulations. In the following, the implementation of a Modelica library for the partitioning of physical models, denoted as MULTIRATE library, was sketched out. Finally, elements of that library were used to partition a detailed robot model and the simulation performance of that partitioned model was compared to “conventional” inline integrators provided by the Dymola tool. This numerical experiment illustrated: a) that the partitioning is feasible also for comprehensive, “real-world” models, and b) that by using a coupling of clocked discretized continuous-time partitions in combination with judiciously selected solver methods considerable simulation speedups can be achieved (the speedup factor was about 2.9 for the example model).

Despite this encouraging result, it also needs to be noted that it can take substantial efforts to find a *good* partitioning and select a combination of solver methods and corresponding integration step sizes that outperform a simulation using “conventional” solvers. Nevertheless, particularly for real-time simulations, investing that additional effort may be worthwhile if real-time constraints cannot be satisfied by using a conventional global solver approach.

## Acknowledgements

Partial financial support of DLR by the German BMBF (within the ITEA2 project MODRIO) and of Dassault Systèmes by the European Union (within the CleanSky project MODELSSA) for this development is highly appreciated.

## References

- [1] Dennis S. Bernstein. The treatment of inputs in real-time digital simulation. *SIMULATION*, 33(2):65–68, 1979.
- [2] Martin Busch. *Zur effizienten Kopplung von Simulationsprogrammen*. PhD thesis, Universität Kassel, 2012.
- [3] Francois Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer Science + Business Media, Inc., 2006.
- [4] Hilding Elmquist, Sven Erik Mattsson, and Hans Olsson. New Methods for Hardware-in-the-Loop-Simulation of Stiff Models. In Martin Otter, Hilding Elmquist, and Peter Fritzson, editors, *2<sup>nd</sup> Int. Modelica Conference*, Oberpfaffenhofen, Germany, March 18–19 2002.
- [5] Hilding Elmquist, Martin Otter, and Françoise E. Cellier. Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems. In *European Simulation Multiconference*, 1995.
- [6] R. Kübler and W. Schiehlen. Modular simulation in multi-body system dynamics. *Multibody System Dynamics*, 4(2-3):107–127, 2000.
- [7] Jim Ledin. *Simulation Engineering*. CMP Books, Lawrence, Kansas 66046, first edition, 2001.
- [8] Modelica Association. Modelica—A Unified Object-Oriented Language for Systems Modeling v3.3. Standard Specification, May 2012. available at <http://www.modelica.org/>.
- [9] Stefan Nowack and Georg Feil. Reduction of time delays in Runge-Kutta integration methods. *SIMULATION*, 48(1):24–27, 1987.
- [10] Martin Otter, Bernhard Thiele, and Hilding Elmquist. A Library for Synchronous Control Systems in Modelica. In Martin Otter and Dirk Zimmer, editors, *9<sup>th</sup> Int. Modelica Conference*, Munich, Germany, September 2012.
- [11] Anton Schiela and Hans Olsson. Mixed-mode Integration for Real-time Simulation. In *Modelica Workshop 2000 Proceedings*, pages 69–75. Lund, Sweden, 2000.
- [12] Tom Schierz and Martin Arnold. Stabilized overlapping modular time integration of coupled differential-algebraic equations. *Applied Numerical Mathematics*, 62(10):1491–1502, 2012.



# Significant Reduction of Validation Efforts for Dynamic Light Functions with FMI for Multi-Domain Integration and Test Platforms

Dr. Stefan-Alexander Schneider                      Johannes Frimberger  
BMW AG, 80788 Munich, Germany  
stefan-alexander.schneider@bmw.de, Johannes.JF.Frimberger@bmw.de

Michael Folie  
IPG Automotive GmbH  
Agnes-Pockels-Bogen 1  
D - 80992 Munich, Germany  
michael.folie@ipg.de

## Abstract

Modern advanced driver assistance systems (ADAS) provide a significant increase in comfort and safety. In many cases, a single vehicle, today, contains more than one assistance system, while the trend to use ADAS continues to grow. At the same time, the number of systems that perform control interventions with safety-relevant functions increases as well. From an overall perspective, it must be assured that the driver assistance systems – individually as well as in the way they interact – function flawlessly in any driving situation and with any driver at the wheel anywhere in the world. This results in an increased development and testing effort for modern ADAS in general. In-development testing based on virtual test driving offers an approach to a solution that allows the validation effort to be significantly reduced while meeting the requirements for safety-relevant functions in the vehicle associated with ISO 26262. This is particularly evident when developing and testing new light functions, which in real-world road tests can often be performed only in conditions of darkness. Dynamic Light Functions are defined as the situation-dependent headlight adjustment consisting of cornering light, headlight leveling and (glare-free) headlight assistance. This paper presents this new methodology.

*Keywords: FMI; AUTOSAR; CarMaker; Sil; Xil; Dynamic Light Function*

## 1 Motivation and Current State of Technology

Today, new light functions are developed in the early stage based on models. These models are subsequently tested for functional performance in the Model-in-the-Loop (MiL) process using simple test cases. Based on requirement specifications, the Simulink models are subsequently migrated to C Code and put into an electronic control unit as an AUTOSAR software component (AUTOSAR-SWC). The approval occurs strictly in the Hardware-in-the-Loop (HiL) test based on real-world measurement data or stimuli. After approval of the software on the HiL-(/SiL-) test rig, testing in the vehicle commences. When using this procedure, though, feedback in the whole vehicle takes place only at a very late point in time. Today, in the Dynamic Light Functions project, dynamic SiL tests are performed based on recorded test drives and manually generated scenarios. Improvement potential still exists here with respect to certain aspects:

1. The open-loop tests, in the case of modified controller software, are reusable only to a limited extent.
2. The transfer to different vehicle configurations or new vehicle variants is possible only to a limited extent. The low availability of prototype vehicles in the various development stages does not permit any tests to be run on the physical prototype.
3. The additional evaluation scenarios which may emerge in the analysis of the interaction of the controller and controller distance have to be recorded in the vehicle test again. The

reproducibility of the scene and the large number of real-world variants is often unfeasible.

By using a vehicle model and corresponding test automation real-world effects can be examined early in the SiL approach. This SiL approach can be integrated into the seamless X-in-the-Loop (XiL) development method [1]. The limited availability of the HiL test rig resource (and the associated late availability of the whole vehicle) could thus at least be partially alleviated.

## 2 FMI Brings Light into Darkness

The Functional Mock-up Interface (FMI) [2] offers the possibility to make use of C Code and Simulink models in a standardized form as so-called Functional Mock-Up Units (FMU) in integration tools. These FMUs can be used for interaction chain tests and design decisions at a very early stage of development. In addition, they are available in the validation chain for dynamic residual bus simulation models. Furthermore, the fast and easy connection of an electronic control unit or an array of ECUs is possible as well.

An FMU can be generated quickly and at low cost using simple means. FMUs consist of a ModelDescription, i.e. a description of the interface (signals and parameters) and a container with a dynamic library for the relevant operating system. Therefore, the generation of such an FMU requires some type of interface specification (e.g. the arxml format [3]) and the associated source code and/or a previously compiled library for the relevant target systems (Figure 1). Even though some tools already offer the possibility to generate FMUs, the objective here is to point out a license-free alternative for quick implementation that can also be used to carry out new requirements with ease.

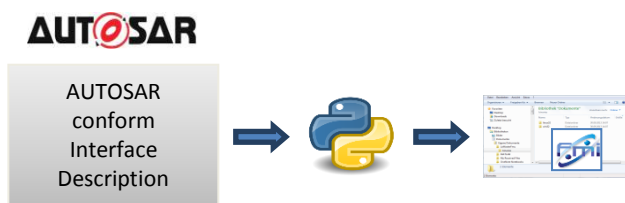


Figure 1: FMU generation.

The ModelDescription is an XML format specified in the FMI standard. All the information required to generate it is available in the AUTOSAR

specification in the form of an arxml description and is therefore retrievable in an automated form. In addition to the interface description, the AUTOSAR XML specification contains the function calls for initialization and the individual calculation steps as well as their step widths. Furthermore, the interface description provides information about the names of the buffers (global variables) for inputs and outputs as well as coding parameters. The FMU interface merely provides an array with all model variables. Consequently, they have to be additionally mapped to the global variables in order to enable the controller to process the current data.

To simplify the integration of the controller into different development environments, a floating-point to fixed-point conversion was directly implemented in the wrapper, again automatically generated from the existing signal description.

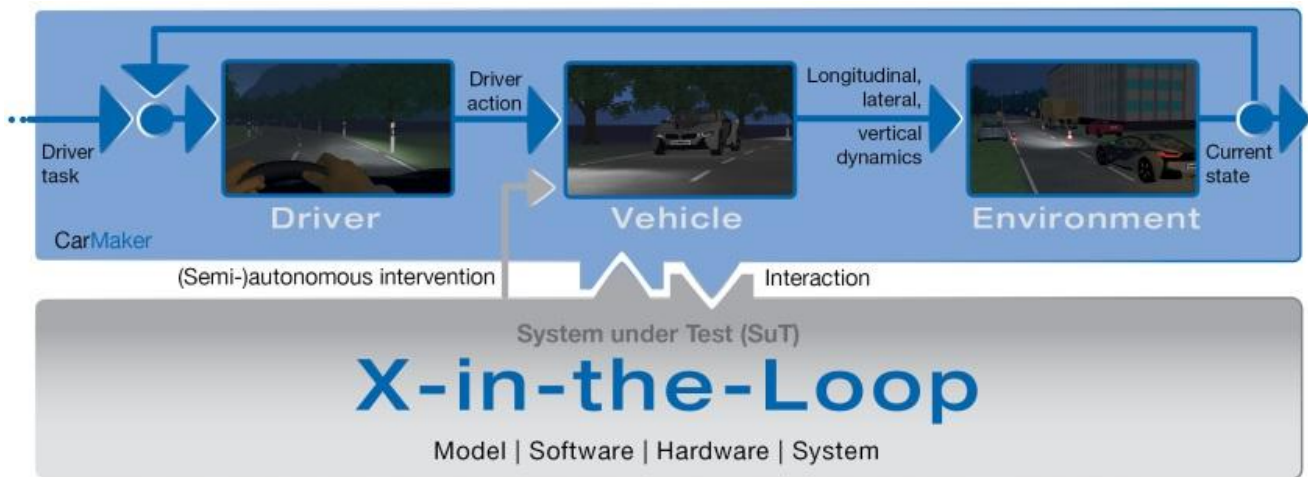
By using this information it is now possible to generate code which initially sets the parameters of the FMU and in the subsequent calculation steps manages the inputs and outputs and calls the actual controller. Together with the controller source code it is possible to compile dynamic libraries which together with the ModelDescription yield a functional FMU.

The FMUs generated in this way can be imported, linked and simulated in various integration tools.

## 3 Consistent Tests through Efficient Integration of Standardized Components

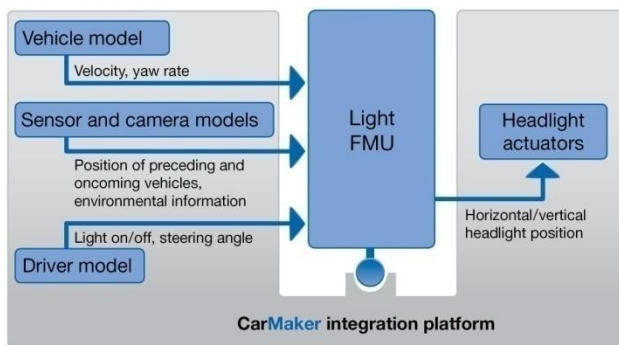
To create the link between the FMUs and the whole vehicle, additional plausible models of vehicle physics and the vehicle environment are necessary. For this purpose, the seamless X-in-the-Loop (XiL) approach was rigorously implemented in CarMaker (Figure 2).

The XiL integration makes it possible to integrate and comprehensively validate all relevant system components, either as models, ECU software or hardware, into the whole virtual and real-world vehicle. As an open integration platform, CarMaker offers interface architecture that is adapted to the vehicle development and in which the FMU approach can be efficiently used. Models, software components and real-world vehicle components are integrated by mouse click into so-called digital prototypes – from the single component through to interlinked (networked) systems.



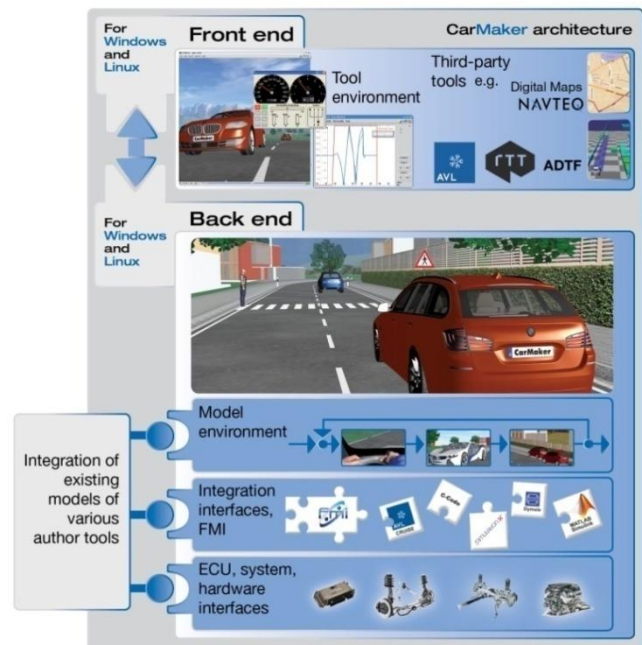
**Figure 2:** X-in-the-Loop enables the early verification and validation of systems. CarMaker offers the appropriate interfaces to integrate all relevant components and systems into a whole virtual vehicle.

Various components such as the powertrain, assistance and control systems etc. as well as instrument and operating concepts may be integrated as needed [4]. The virtual integration creates the prerequisites to check what effects the tested components have on the overall performance of the whole vehicle. Unfavorable design decisions and even functional faults can thus be detected earlier and allow corresponding design decisions to be optimized [5]. The digital prototypes are verified as a total system in virtual test driving and serve as input for our light assistance FMU (Figure 3).



**Figure 3:** Interfacevariables to implement the light FMU into the CarMaker integration platform.

In practical terms, we receive signals from vehicle dynamics, additional environment sensors and camera systems as well as driver inputs such as steering angle or manual headlight requests. In addition to the powerful vehicle and driver model, CarMaker encompasses a complete environment simulation, consisting of roads with corresponding static traffic objects and moving traffic as well as environment sensors and digital maps (such as NAVTEQ or Google Earth). As a result, the test environment is modeled at a high level of realism, see Figure 4.



**Figure 4:** Function models and software components can be integrated into the integration environment either as Functional Mock-Up Units or hardware components. In addition, CarMaker allows navigation systems and software development tools to be connected as well.

Furthermore, as a test platform, CarMaker offers a maneuver description that is based on the principles of real-world road tests. Complex open- and closed-loop tests are carried out as maneuver instructions. The virtual test drive is reproducible and can be easily modified as needed. This makes it possible to evaluate new developments in virtual test driving and to tangibly experience them in realistic driving situations. The test cases for the light functions are migrated from test driving into the CarMaker language and carried out in the TestManager (Figure 5).

Item Description	Par1	Par2	Par3	Par4	Criteria	Res.Date	Result
<b>Global Settings</b>							
<b>Driving_Ser</b>							
Velocity							
Variation 1	30					15:25:32	●
Variation 2	50					15:25:41	●
Variation 3	30					15:25:44	●
Variation 4	120					15:25:44	●
<b>Steer_Ser</b>							
Velocity		Steeringangle					
Variation 1	30	15				15:25:49	●
Variation 2	50	10				15:25:54	●
Variation 3	30	3				15:25:56	●
<b>Backwards_Ser</b>							
Velocity		Steeringangle					
Variation 1	5	25				15:25:59	●
Variation 2	10	20				15:26:01	●
Variation 3	20	10				15:26:03	●

Figure 5: Test cases from real test driving.

By combining in-development testing, the standardized FMI and the multi-domain integration and test platform the development and test effort can be significantly reduced. A maneuver catalog that has been defined once is thus available for design, verification and validation purposes in all stages of the development process. The maneuver catalog is specifically extended according to the development stage in order to comprehensively verify and validate the design and implementation of the ECU software. This allows fast, convenient exchanges and testing of vehicle components as FMUs, both within the BMW organization and in collaboration with suppliers. In other words, the OEM and the suppliers are able to use the same models and functions for architectural decisions and virtual test driving.

## 4 Use in the Development Process

Last but not least, this is where the advantages of virtual test driving prove their viability (Figure 5): providing test scenarios (through to complex customer use cases) which can be modified as needed and are reproducible on the one hand and reproducible test results on the other.

In the development process, FMUs can be used in an SiL environment at an early development stage. Software modifications and their effects on existing test cases can be easily realized and tested due to the standardized components.

Aside from this range of applications, FMUs can also be used as substitutes for electronic control units not currently in existence in an array or composite of ECUs together with existing, real-world ECUs. This way, they help to make tests on the whole system possible at an early development stage.

In the case of the dynamic light function, in addition to numerous vehicle dynamics parameters provided by CarMaker, camera-based information is processed as well. This information is either provided by virtual sensors or – ideally – by the real ECU of the camera. To enable the ECU to actually deliver realistic values, virtual driving scenes including other traffic from CarMaker are captured and processed by the camera in a darkened room. This means that the virtual test drive is filmed by the real-world vehicle camera in order to run a so-called interaction chain test. The camera is used to detect the presence of other vehicles within the blinding range of the light. This information is forwarded to the light function in order to activate the glare-free range of the high beam (Figure 6).

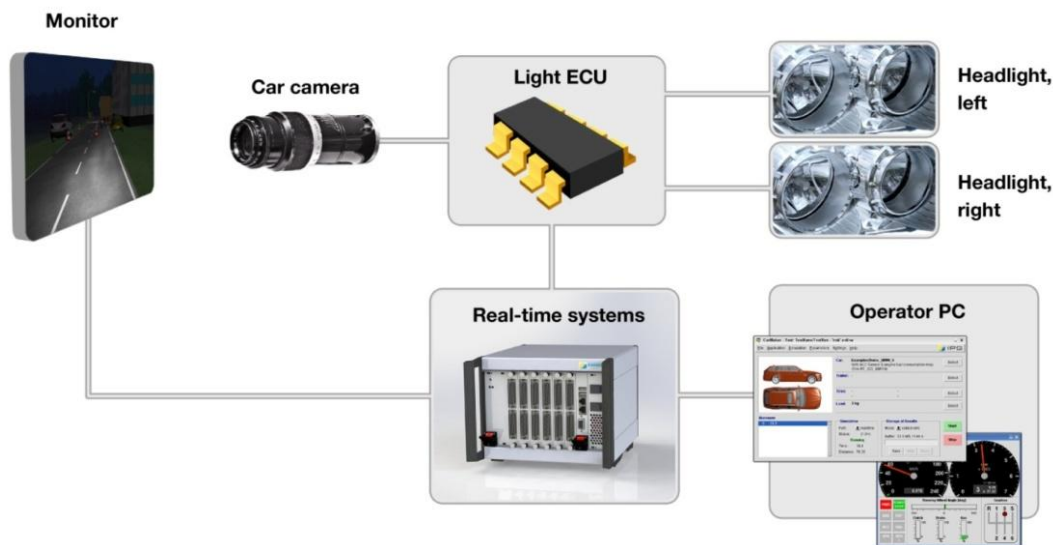


Figure 6: Set-up of an HiL test rig for the event chain test: Operator PC and HiL system, real-world or simulated controls for the light, light ECU, headlights with headlight actuators, camera with integrated ECU and monitor for filming.

This adds significant value compared to the tests that have been run up to now, in which only the ECU could be served with a residual bus simulation but the specific signals and functions of the camera could not be used.

Linking real-world headlights with the environment described above now represents the next step. This allows the entire interaction chain, from the camera through to the actuator, to be modeled and tested. As a result, detailed statements about the functional performance of the whole system can already be made on the test rig, which makes it possible to considerably reduce the validation effort on the real-world vehicle. Figure 7 shows a snapshot from a video [6] for comparison of a real-world and a corresponding virtual test driving situation.



**Figure 7:** Comparison of the light function: video with glare-free high beam and its simulation.

## 5 Conclusions

By using both the standardized interface specification FMI and an integration and test platform (from MiL, to SiL and through to HiL) the current development process can be made significantly more efficient and the existing gap in the area of the SiL tests can be closed. In such a development environment the presented control loop consisting of the “the car on the road” and “light events detected by the camera” can be studied in detail. The control loop of interest can then be extended step by step, e.g. additionally including the position of the headlight actuators, and/or the simulated illumination of the headlights. A reduction of the testing effort associated with the large number

of variants can be achieved by the DoE (Design of Experiment) method in combination with carefully selected real-world road tests [7]. For this purpose, critical corner case parameters can simultaneously be tested in the design space to be validated and, in parallel, simulated in virtual test driving. The models on which these tests are based are validated. Proceeding from this database, additional real-world road tests could be made superfluous by the continuing validation being supported by the results thus obtained. This harbors further remarkable potential for reducing the validation effort and, ultimately, controlling this effort as well.

## References

- [1] Schick, B.: Mission V-Process enhancement by integrated vehicle performance evaluation within an entire X-in-the-Loop process, Keynote SIAT ARAI 2013
- [2] FMI Development Group: FMI – The Functional Mock-up Interface. [\[https://www.fmi-standard.org/\]](https://www.fmi-standard.org/): [January 21, 2014]
- [3] AUTOSAR Development Partnership: AUTOSAR, [\[http://www.autosar.org/\]](http://www.autosar.org/): [January 21, 2014]
- [4] Martinus, M., Deicke, M. and Folie, M.: Virtual Test Driving: Hardware-Independent Integration of Series Software, ATZ Elektronik 10/2013
- [5] Schneider, S.-A., B. Schick and H. Palm: Virtualization, Integration and Simulation in the Context of Vehicle Systems Engineering, embedded world 2012, Nuremberg 2012
- [6] BMW UK: BMW Intelligent Headlight Technology: Long Version: [\[http://www.youtube.com/watch?v=dvPZ3H1Vm4\]](http://www.youtube.com/watch?v=dvPZ3H1Vm4): [January 21, 2014]
- [7] Palm, H., Holzmann, J., Schneider, S.-A., Kögeler, H.M. and Pfister, F.: The Future of Car Design: Optimisation with Systems Engineering, ATZ Elektronik 06/2013





# Hardware-in-the-Loop (HIL) Simulation with Modelica - A Design Tool for Thermal Management Systems

Sidney Baltzer   Thomas Lichius   Jörg Gissing   Peter Jeck   Lutz Eckstein  
Institute for Automotive Engineering (ika) – RWTH Aachen University  
Steinbachstraße 7, 52074 Aachen  
[baltzer@ika.rwth-aachen.de](mailto:baltzer@ika.rwth-aachen.de)

Jörg Küfen  
Forschungsgesellschaft Kraftfahrwesen mbH  
Steinbachstraße 7, 52074 Aachen  
[kuefen@fka.de](mailto:kuefen@fka.de)

## Abstract

Due to the higher complexity of electrified vehicles the requirements for vehicle components and vehicle design augment and new development tools are desirable. The following paper describes the design of a hardware-in-the-loop test bench along with its structure using Modelica and a Remote Process Communication library. The aim is to support the development of components and operational strategies under realistic boundary conditions illustrated by the example of a waste heat recovery system. The test bench is planned and built up within the scope of the public founded project qOpt at the Institute for Automotive Engineering (RWTH Aachen University) in cooperation with the Forschungsgesellschaft Kraftfahrwesen mbH Aachen and the Institute of Automatic Control (RWTH Aachen University).

*Keywords: hardware-in-the-loop simulation, thermal management; vehicle simulation; combined heat and power generation, waste heat recovery, electric vehicle, Plug-in hybrid vehicle, thermal storage*

## 1 Introduction

A pursued objective of politics and industry is to improve the efficiency and reduce the emission of individual mobility. For achieving that, the electrification of the drive train seems to be a promising approach. The wide distribution of purely electric vehicles lacks due to their short driving ranges since the specific energy content of current traction battery systems is rather low which leads to a high vehicle

weight. Besides, the costs for such systems are still quite high. Plug-in hybrid electric vehicles (PHEV) or range extended electric vehicle (REEV) provide the opportunity to combine the advantages of a conventionally propelled vehicle such as their high driving ranges with the possibility of driving electrically and thus without emissions. To increase the electric driving range an efficient treatment of the electric energy is obligatory. This includes the optimization of the drive train, the reduction of the electric energy demand for auxiliary consumers for example by means of an intelligent thermal management. In order to exploit the maximum potential of such a drive train a complex operational strategy in consideration of all energy forms has to be provided. For example in the scope of the public founded project qOpt, which enables this research, especially the reduction of auxiliary electric heaters during the winter term is focused. Therefore a waste heat recovery system in form of a latent heat storage in combination with an operational strategy will be developed.

For an a priori design the requirements for new simulation tools augment. But often not every component may be simulated properly so hardware tests are still necessary. A combination of simulation models and specific hardware components in a hardware-in-the-loop (HIL) environment provides the possibility to reduce building up physical prototypes for a high number of variations.

HIL systems are widely used for control systems, like engine control units. An extension of such systems for prototyping components is the logical consequence and is to be considered further on in this paper.

## 2 Hardware-in-the-Loop System

In general, HIL simulation systems provide the opportunity for the following three points:

- Component tests and component design
- Design, test and validation of operational strategies
- Control of test bench components

Such a system is built up at the Institute for Automotive Engineering (ika), RWTH Aachen University, in cooperation with the Forschungsgesellschaft Kraftfahrwesen mbH Aachen (fka).

The main system consists of the simulation environment, the test bench components and its controls and the data exchange process. All three systems are explained further in the following sub chapters.

### 2.1 Holistic Model Library

For the mentioned reasons a holistic model library has been developed at the ika in cooperation with the fka and is constantly increased and improved [1].

The library contains different kinds of vehicle models, including their drive train, passenger cabin, and their respective cooling circuits. Also building systems may be considered, to solve future problem issues like vehicle-to-home (V2H) or vehicle-to-grid (V2G) applications. All models can be controlled and evaluated under different dynamic boundary conditions (e.g. drive cycles, ambient conditions) (cf. Fig. 1).

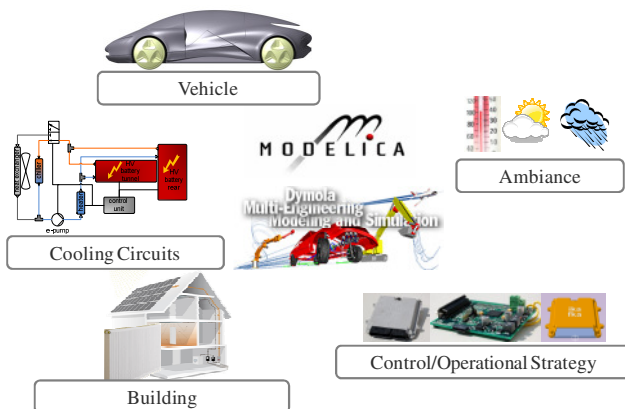


Fig. 1: Holistic simulation approach

The library is implemented following an approach of high scalability and modularity, so the level of detail may be adjusted depending on the issue to be investigated (cf. [1]).

### 2.2 Thermo-hydraulic test bench

The common interfaces for thermo-hydraulic simulations are the temperature, volume flow rate and the pressure. When emulating physical systems in a simulation environment the respective physical values have to be provided with a thermo-hydraulic test bench at every time step. The used test bench for the HIL system is shown in Fig. 2. For the hydraulic part, a controllable fluid pump and several controllable valves are integrated to adjust the volume flow rate and the relative pressure at the device under test (DUT).

The temperature is regulated with a heating device and a fluid cooling system. When connecting a refrigerant system also temperatures less than ambient temperature can be achieved.

The test bench is operated by a CompactRIO system. Since the system is not hardly real time capable with bigger models also a PXI System may be connected, which is presented in [3].

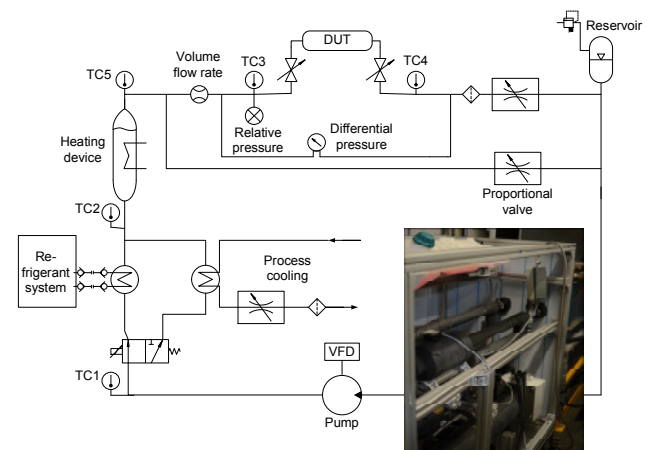


Fig. 2: Thermo-hydraulic test bench

### 2.3 Data Exchange Process

The software configuration of the HIL consists of several applications, which are simultaneously working together. The participating applications address different concerns at the HIL, e.g. simulation and test bench control.

Proper execution and interaction of all applications need to be assured. Thereby the resulting challenge for the HIL system is the elimination of unwanted side effects between the applications in order to avoid mutual interference of the applications during operation.

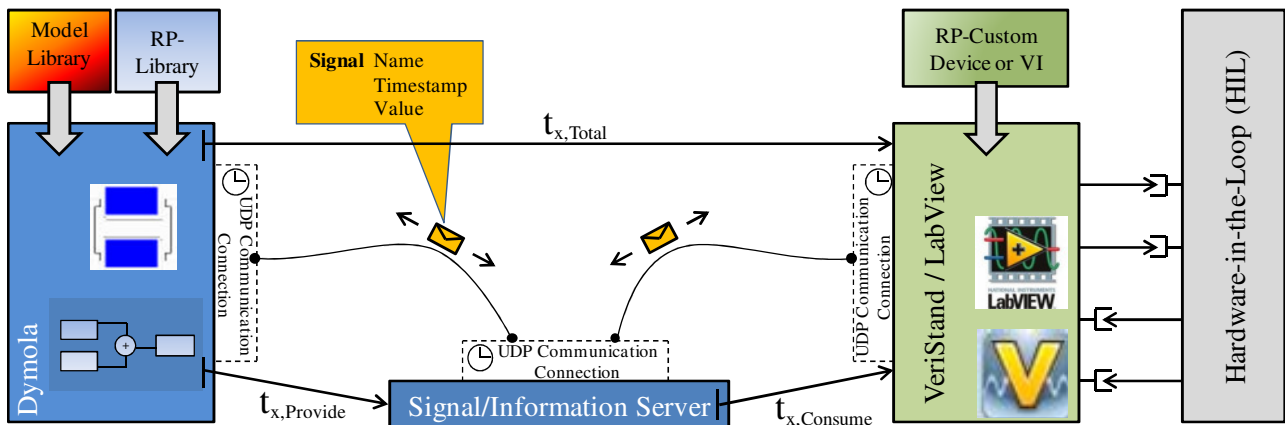


Fig. 3: Software structure of the HIL system. Multiple applications are joined for operation.

An overview of the participated applications currently in use at the HIL is shown in Fig. 3. Most important are Dymola and LabView. Dymola is responsible for hosting the physical model of the surrounding environment, which generates control requests for the test bench and the DUT. The management of the HIL system itself is based on National Instruments LabView [6] in optional combination with VeriStand [7], executed on a CompactRIO hardware and a Windows based personal computer. The CompactRIO executes all requests coming from the physical model output. LabView in combination with the CompactRIO ensures reliable control of the hardware of the test bench and keeps the test bench within applicable operating conditions.

At the same time LabView and the CompactRIO capture measurement data from test bench and DUT and return these back to the physical simulation in the Dymola environment. As Dymola is normally not targeted for real-time interaction with control components, new coupling features have been integrated to Dymola in order to implement the previously mentioned linkage of applications.

The connection between the applications is realized by an additional Remote Process Communication (RPCom) interface library, developed at fka. The library provides communication and synchronization elements, which are added to the physical model in Dymola in order to build an externally accessible interface with input and output data (cf. Fig. 4). This interface is accessible while the simulation is running. Corresponding elements of the RPCom Library are as well integrated in a LabView VI<sup>1</sup> or in VeriStand using a RPCom Custom Device [8].

<sup>1</sup> Individual, decoupled operation of all applications can be performed even if the RPCom elements have been integrated, allowing enhancements of the physical model and the test bench in parallel without removing the RPCom elements.

The RPCom library is implemented in .NET and embedded by Dymola using a system native c wrapper [5]. To allow access to the .NET components by the system native functions called by Dymola the intermediate code is modified such that all managed code elements are exported with system native interfaces [9]. The RPCom functionality can be used to even extend the co-operation of tools at the HIL to further applications, like e.g. a driving simulator, if these are required.

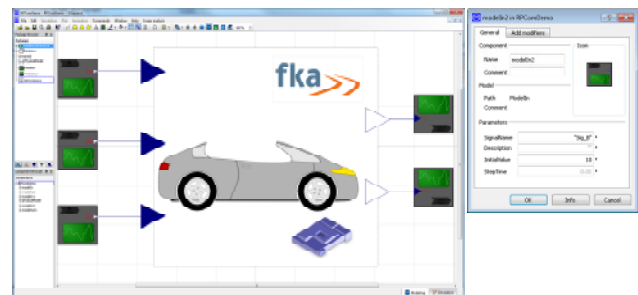


Fig. 4: Simple physical model in Dymola with additional RPCom elements

Main responsibility of the RPCom library is binding the individual applications together by organizing a coordinated signal exchange. This is realized by the RPCom library using UDP communication making distributed execution of all applications on a multi-computer network possible, completely decoupling the control of the test bench hardware from the simulation itself. Supplementary to data exchange, a synchronization functionality for all applications, necessary to make the operations of simulation and test bench behaviors coherent, is taken out by the RPCom library.

The exchange of signals between the applications is organized by symbolic IDs. Finally this means that all signals are managed within a signal pool by a unique symbolic name, which is associated with in-

formation on the corresponding value along with a the timestamp of last known validity of the value and supplementary meta-data, like physical unit or a detailed signal description. Currently, all elementary data types, such as double, integer or boolean values are supported. Vectors shall be supported in a future extension of RPCoM.

All signals are initially stored in local caches assigned to each participating application. In regular, configurable cycles - which depend on the timing requirements - these caches are synchronized. The synchronization can be selected to match the individual timing requirements of the application, avoiding unnecessary data exchanges. Altogether the synchronized signal caches of the applications build up a distributed information server.

For reliable HIL operation, the connection of applications and the timing behavior realizable by the RPCoM library are relevant in order to ensure deterministic HIL operations.

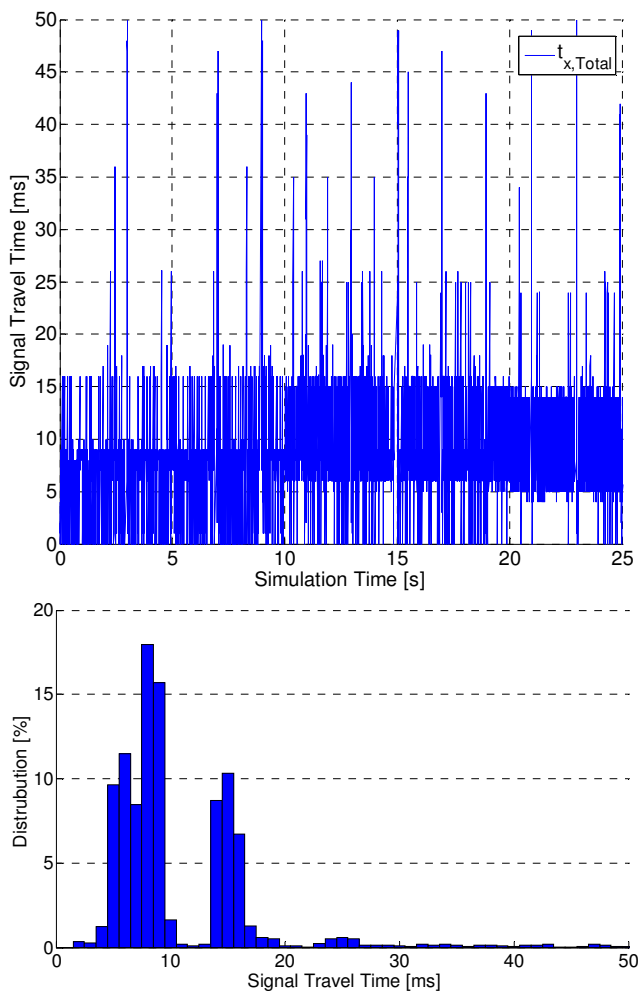


Fig. 5: Signal travel time ( $t_{x,Total}$ ) and distribution measured during HIL operation for exchanged data between the coupled applications. Configured cycle time is 10 ms.

Due to the systems thermal inertia, soft hardware requirements can be applied for the cooperation.

Fig. 5 shows a timing measurement executed on the HIL system with a configured synchronization time of the application caches of 10 ms by the RPCoM library. In the analyzed scenario Dymola and LabView are operated on the same machine. A pool of 25 signals is exchanged between the applications. The cycle time of the Dymola Model is 100 ms while HIL cycle time for test bench control is set to 10 ms. As the measurements in Fig. 5 implies, the exchange rate is very stable around 10 to 20 ms.

The benefit of the integration of the RPCoM library is to distribute different HIL concerns to multiple computer systems within a network, e.g. decoupling hard real-time from weak real-time requirements. Also an easy partitioning of different Dymola model segments to more than one computer system can be realized, allowing integration of even complex and computation time intensive model configurations. Both benefits are utilized within the HIL system.

### 3 Augmented CHP usage of the internal combustion engine in electrified vehicles

Especially in winter terms the thermal management of electrified vehicles represents a major challenge. A temperature sensible component e.g. the traction battery needs to be conditioned and heating energy has to be provided for the passenger cabin. When driving purely electrically the heating energy has to be supported electrically which directly affects the electric driving range. Thus, waste heat recovery is a promising approach.

In the scope of the project qOpt the electrified vehicle Opel Corsa Hybrid 3 (cf. [2]) of ika is converted to a Plug-in-hybrid electric vehicle. Besides, an optimization of the thermal management of the vehicle is considered. In this article the potential of the integration of a thermal storage into a PHEV to enhance the electric driving range is further analyzed by means of hardware-in-the-loop simulations.

#### 3.1 System architecture

The vehicle data including the passenger's cabin are listed in Tab. 1. Since the internal combustion engine (ICE) has a high potential for waste heat re-

covery, the ICE and its cooling circuit is considered as the device under test (DUT).

Opel Corsa Hybrid	
empty weight	1150 kg
power (ICE)	44 kW
power (electric machine)	37 kW
battery capacity	16 kWh
cabin volume	3 m <sup>3</sup>
window surface	2 m <sup>2</sup>

Tab. 1: Vehicle data of the Opel Corsa Hybrid 3

In Fig. 6 the schematic hardware setup is shown. An additional heat exchanger (HEX) is integrated into the cooling circuit of the ICE to provide the possibility to warm up the ICE or use its waste heat. The heat exchanger is the interface between both the electrified vehicle and the HIL system. Since the integrated pump is belt driven, an additional electric pump is integrated to achieve higher volume flow rates when driving at low engine speed or when driving purely electrically. Furthermore to monitor and control the cooling circuit, different sensors for volume flow rate, relative pressure and temperatures are integrated.

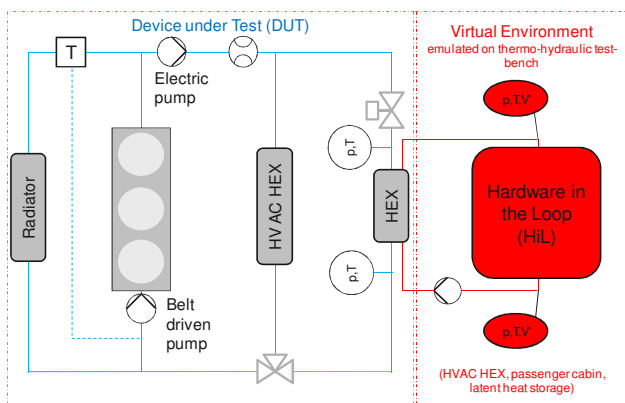


Fig. 6: Schematic view of the HIL setup and its connection to the cooling circuit of the internal combustion engine

To analyze the waste heat at different driving cycles, the vehicle is placed on a dynamic chassis dynamometer. The driving resistance for the specific Opel Corsa are adjusted, so a realistic power output is achieved. The current test setup is shown in Fig. 7. As mentioned above a winter term scenario and the possibility of waste heat recovery is analyzed. But since with this setup the winter term boundary conditions can only be adjusted to the ICE it is necessary to simulate the remaining vehicle components. For

this purpose inter alia the library mentioned in chapter 2 is used.

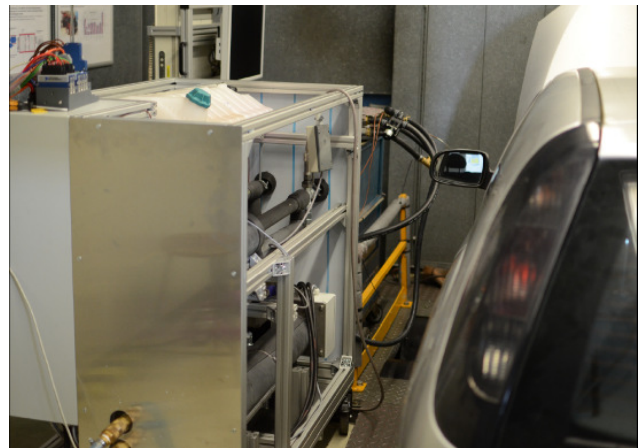


Fig. 7: System setup with the Opel Corsa on the dynamic chassis dynamometer connected to the HIL test bench

### 3.2 Simulation models

In the application case all thermal energy sinks are modeled in Dymola/Modelica. This includes the model of the passenger cabin and the HVAC unit. Besides, a thermal storage is considered. The heat exchanger of the HVAC uses the measured temperature to calculate the heat flow rate into the cabin. The heating demand is determined by the passenger cabin model. It includes the different convective and radiative heat flow rates to and from the environment (cf. Fig. 8 and [4]). Inside the heating circuit an additional electric heater is installed to provide heating energy when the vehicle is driven purely electrically.

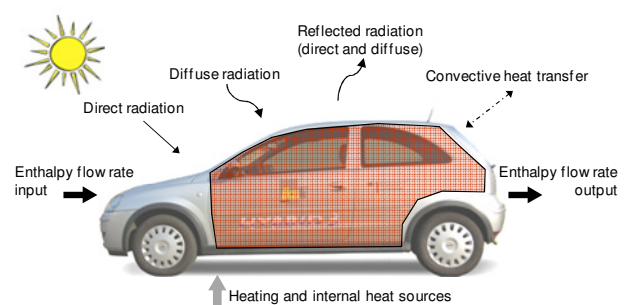


Fig. 8: Energy flows in passenger's cabin model

In this application example a latent heat storage was chosen. It is modeled according to [10]. The latent material has a melting temperature of about 65°C. Depending on the temperature gradient between the cooling fluid and the latent material the thermal conductance is calculated.

### 3.3 Scenario and boundary conditions

To analyze the benefit of an advanced combined heat and power (CHP) usage of the ICE (engine data cf. Tab. 1) a realistic scenario is defined. A drive cycle with a highway part and a rural part is used for the simulations. On the highway the vehicle is propelled by the internal combustion engine at a constant speed of 110 km/h. The waste heat is used to warm up the ICE and to provide heat for the passenger's cabin. Shortly before the thermostatic valve opens (engine is warmed up) a valve is operated, opening the bypass to the thermal storage. In this way an advanced combined heat and power generation usage of the ICE is achieved since the opening time of the thermostatic valve is minimized. Thus, less heat is rejected to the environment and a higher amount of waste heat is used. As mentioned before, all heat sinks are simulated in Modelica and the respective heat flows rates are transferred from the engine's cooling circuit in the thermo-hydraulic test bench.

Subsequently, the vehicle enters a rural zone (Urban part of HYZEM cycle) and is operated purely electrically. The stored energy from the thermal storage is then used for providing heating energy for the passenger's cabin, so the high value electric energy need not to be used for heating purposes.

As ambient conditions a typical winter scenario is chosen (0°C ambient temperature, 100 W/m<sup>2</sup> solar radiation) (cf. [11]).

## 4 Results

In Fig. 9 the dynamic profile of the engine cooling temperatures before and after the heat exchanger that is connected to the HIL are shown (cf. Fig. 6).

The initial temperature of the engine is 10 °C. The waste heat of the ICE warms up the engine and by the means of the heat exchanger in the HVAC the passenger cabin. At the beginning only little heating energy is transferred through the HVAC heat exchanger because of the low temperature gradient. At a temperature of 84 °C in the ICE cooling circuit the valve, to regulate the volumetric flow through the thermal storage is opened. Due to the high temperature gradient a high amount of heat is transferred to the thermal storage. The measurement clearly shows this point. Both the temperature before and after the heat exchanger are reduced. Subsequently the temperature gradient and therefore the heat flow rate into the storage decreases (cf. Fig. 9).

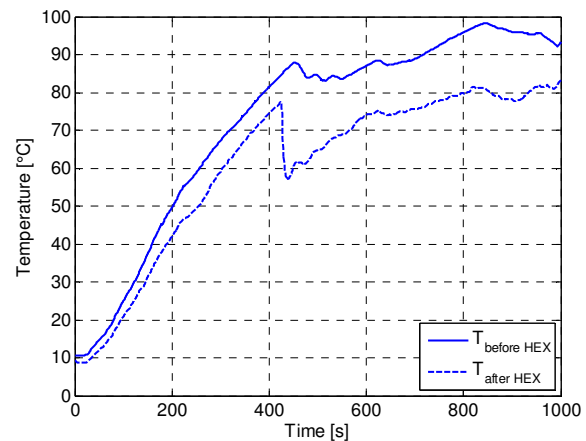


Fig. 9: Engine cooling temperatures before and after heat exchanger (cf. Fig. 6)

Fig. 10 shows the dynamic curve of the heat flow rate to the cabin and to the storage. Within the first 300 seconds the heating power increases since the engine cooling temperature increases. After that the control unit for cabin heating demands a lower heating power to maintain comfort temperature (cf.[4]).

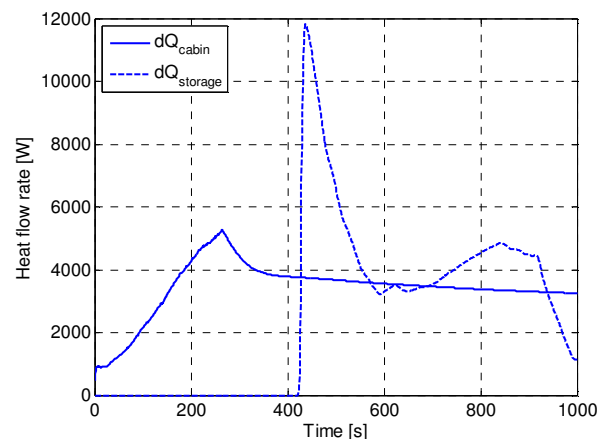


Fig. 10: Heat flow rates for the passenger's cabin heating and to the PCM storage (cf. Fig. 6)

In Fig. 11 the temperature curve of the thermal heat storage is shown. The melting point of the latent material is clearly stated out. At the end of the ride the thermal storage reaches a temperature of about 85 °C. The contented energy amounts about 700 Wh.

During the subsequent purely electric ride the heating energy may be used for cabin heating and by this the heating demand for the next 600 seconds may be provided nearly completely by the stored waste heat of the combustion engine. Thus, the electric driving range can be enhanced by about 4 km.

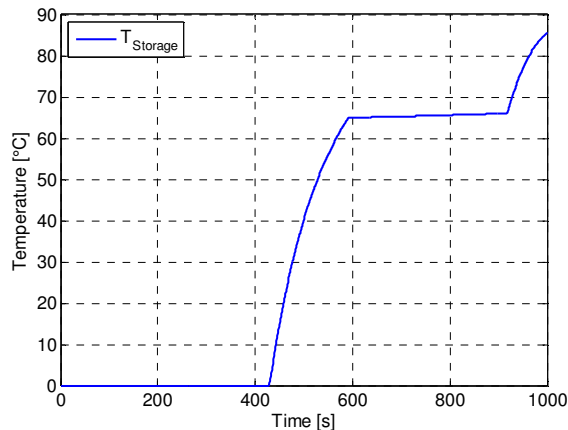


Fig. 11: time dependent temperature curve of the latent heat storage

## 5 Conclusion and Outlook

Due to the higher complexity of electrified vehicles the requirements for vehicle components and vehicle design augment and new development tools are desirable. In this article a HIL system using Modelica and an UDP interface has been presented which was developed at the Institute for Automotive Engineering (ika), RWTH Aachen University, in cooperation with the Forschungsgesellschaft Kraftfahrwesen mbH Aachen (fka). The connection between the applications is realized by an additional Remote Process Communication (RCom) interface library, developed at fka.

Performance tests show that the connection of applications and the timing behavior is reliable to ensure deterministic HIL operations, at least for low real time requirements. Besides, by this data exchange progress it is possible to separate the simulation and the test bench control hardware, especially when models with a higher complexity demand high performance hardware.

An application example was given in which the potential of integrating a thermal heat storage unit into the cooling circuit of the ICE of a PHEV was investigated by means of HIL simulation. In order to measure the usable amount of waste heat of the ICE the vehicle was placed on a dynamic chassis dynamometer. All heat sinks, like the passenger cabin as well as the thermal storage were simulated in Dymola and the respective physical values were transferred to the thermo-hydraulic test bench. The results show that a reasonable amount of waste heat could be recovered in a thermal storage. Therefore, the electric driving range can be enhanced by provid-

ing the heat energy of the passenger cabin by the thermal storage unit.

The HIL system will be used further in the project qOpt to develop an operational strategy in consideration of the heat demand for electrified vehicles taking into account a latent heat storage system. Besides, in another project an electrical and thermal coupling between vehicles and buildings will be investigated further on.

## 6 Acknowledgement

The authors would like to thank the State of North Rhine-Westphalia/EFRE for their financial support in the research project qOpt.

## References

- [1] Bouvy C., Baltzer S., Jeck P., Lichius T., Gissing J., Eckstein L.: Holistic Vehicle Simulation using Modelica - An Application on Thermal Management and Operational Strategy for Electrified Vehicles. In: Proceedings of the 9th Modelica Conference 2012, Oberpfaffenhofen, Germany, Modelica Association, 3-5 September 2012.
- [2] Biermann J.-W., Gnörich B.: From Micro to Full Hybrid - Two cars developed by fka and ika. In: International Automotive Congress 2008, Shenyang, China, 31 October 2008.
- [3] Baltzer S., Lichius T., Gissing J., Jeck P., Küfen J., Barkow A., Eckstein L.: Hardware-in-the-Loop-Prüfstand zur Auslegung von Thermomanagementsystemen im PKW. In: Virtuelle Instrumente in der Praxis 2013. Mess-, Steuer-, Regel- und Embedded-Systeme - Begleitband zum 18. VIP-Kongress, Germany, 2013.
- [4] Bouvy C., Gissing J., Lichius T.: Kühlsystementwicklung und Wärmemanagement für PlugIn-Hybridfahrzeuge. In: Proceedings of the E-MOTIVE Expertenforum „Elektrische Fahrzeugantriebe“, Aachen, Germany, 7-8 September 2011.
- [5] Modelica Association (2013): Modelica Language Specification 3.2, [www.modelica.org](http://www.modelica.org).
- [6] National Instruments Labview, User Manual [www.labview.com](http://www.labview.com).
- [7] National Instruments Veristand, User Manual [www.veristand.com](http://www.veristand.com).
- [8] National Instruments – Customizing the NI VeriStand Engine, NI Whitepaper 12640, 9th August 2013. [www.ni.com](http://www.ni.com).

- [9] Daskin D. - Simple Method of DLL Export without C++/CLI, CodeProject 27th June 2009. [www.codeproject.com](http://www.codeproject.com).
- [10] Mehling H., Cabeza L. F.: Heat and cold storage with PCM. An up to date introduction into basics and applications. Berlin, Germany, 2008.
- [11] Strupp N. C., Lemke N.,: Klimatische Daten und Pkw-Nutzung: Klimadaten und Nutzungsverhalten zu Auslegung, Versuch und Simulation an Kraftfahrzeug-Kälte-/Heizanlagen in Europa, USA, China und Indien. Frankfurt a. Main, Germany, 2009.



# Integrated Vehicle Thermal Management in Modelica: Overview and Applications

John Batteh<sup>1</sup>

Jesse Gohl<sup>1</sup>

Sureshkumar Chandrasekar<sup>2</sup>

<sup>1</sup>Modelon, Inc.  
Ann Arbor, MI  
United States

<sup>2</sup>Modelon, Inc.  
Hartford, CT  
United States

[john.batteh@modelon.com](mailto:john.batteh@modelon.com)

[jesse.gohl@modelon.com](mailto:jesse.gohl@modelon.com)

[chandrasekar.sureshkumar@modelon.com](mailto:chandrasekar.sureshkumar@modelon.com)

## Abstract

This paper highlights the use of a coordinated suite of Modelica libraries for vehicle thermal management applications. The models are implemented using the Vehicle Dynamics Library, Liquid Cooling Library, and Heat Exchanger Library from Modelon. An integrated vehicle thermal management model is implemented, including the key physical and controls models. The model is used to highlight complex, multi-domain interactions between the physical and control systems over drive cycles for combined thermal and fuel efficiency studies. The model is also used to support controller development and optimization as an FMU integrated into Simulink. The flexibility of FMI-based workflows is also illustrated via batch and Monte Carlo simulations in Excel. A heat exchanger application coupling inputs from CFD illustrates the use of higher fidelity models from Heat Exchanger Library for calculation of performance degradation due to non-uniformity.

*Keywords: vehicle thermal management; thermal systems; fluid systems; vehicle modeling; powertrain; engine; transmission; controls; Simulink; FMI*

## 1 Introduction

To meet increasingly stringent fuel economy and emissions standards, automotive original equipment manufacturers (OEMs) and suppliers have sought novel technologies to increase fuel efficiency. Given the complexity of vehicle systems, the need for increasingly sophisticated analytic tools to perform concept evaluation, capture multi-domain system interactions, and develop and validate control strategies grows. A Modelica-based platform for simulation of vehicle systems is ideal for this type of work as it naturally captures multi-domain interactions, allows flexibility in model complexity to support a

range of applications, and supports integration of physical and control systems.

Traditional vehicle models for fuel consumption have often ignored thermal effects as the inclusion of thermal effects increases the model development and parameterization effort and can affect computational efficiency for models which are run on drive cycles which can be thousands of seconds in duration. However, in the search for fuel efficiency gains, advanced technology is rapidly advancing from research and development to production, including the associated controls development. Due to the complex interactions between vehicle subsystems, vehicle thermal management (VTM) requires a holistic approach to minimize energy consumption without violating thermal limits for the various systems. Previous work in Modelica has highlighted this need for thermal management of electrified vehicles [1].

A real world example of a vehicle level technology that has the potential for fuel economy gains, but with a direct impact on thermal management is grill shutters. Grill shutters can be used to restrict or completely close airflow to the front of the vehicle. While this reduction in airflow positively impacts fuel efficiency by reduction in aerodynamic drag, reduced airflow through the front end of the vehicle degrades cooling capacity through the heat exchanger stack which contains the radiator, condenser, and potentially additional coolers such as oil and charge air coolers. Grill shutters can be mechanical or potentially even actively controlled. Clearly attribute balancing at the vehicle level is required to manage fuel consumption gains and the cooling requirements for the various fluid systems over a range of driving conditions and ambient environments. Active grill shutters can be found in production on a range of vehicles from various manufacturers, including the Ford Focus, Chevrolet Cruze, Cadillac ATS, Dodge Dart, and Ram 1500 [1].

To illustrate the complex interactions between vehicle subsystems, vehicle controls, and thermal

controls, an integrated vehicle thermal management model is implemented with grill shutters. The models are implemented using a coordinated suite of Modelica libraries from Modelon. The models are implemented using the Vehicle Dynamics Library (VDL) [3], Liquid Cooling Library (LCL) [4], and Heat Exchanger Library (HXL) [5]. The model captures vehicle dynamics on drive cycles including key thermal dynamics. The model also includes critical vehicle controls and thermal controls for the grill shutters and fan. This paper provides an overview of the model, key subsystem implementations, and key features of the various libraries to support this type of modeling.

The integrated vehicle thermal management model is used as a demonstrator for Modelica-based workflows for illustrative controls development and robustness applications. The controls development application illustrates the implementation and optimization of an active grill shutter strategy. The Modelica VTM model is exported as an FMU via Functional Mock-up Interface [6] for integration in Simulink with the grill shutter and fan controls. Integration in Simulink is via FMI Toolbox for MATLAB [7] from Modelon. DOE techniques are used to optimize the combined grill shutter and fan settings for minimum fuel consumption. A sample optimized controller is implemented to show the fuel economy impact on several different drive cycles. Sample robustness studies are also performed with the VTM FMU in Excel via FMI Add-in for Excel [8] from Modelon. These applications include batch simulations and Monte Carlo simulations and also illustrate some of the scripting capabilities in FMI Add-in for Excel. Workflows that involve coupling higher fidelity models from Heat Exchanger Library with CFD input data for heat exchanger performance are shown.

This paper describes a coordinated suite of libraries for vehicle thermal management, an integrated model including grill shutters, and applications of this model for controls development and robustness using FMI. These examples illustrate the multi-domain approach needed for vehicle thermal management applications. Given the importance of model deployment outside of traditional CAE environments to support model-based systems engineering, workflow aspects via FMI are highlighted.

## 2 Integrated VTM Model

This section outlines the key multi-domain component and subsystem models in the integrated VTM model that support the subsequent applications. The

main model components and subsystems are detailed. The model shown is representative of system level models for use in thermal system design and performance characterization with parameterization as a demonstrator model and thus does not include any customer-proprietary data (future publications will highlight industrial models and applications pending publication approval). The vehicle parameterization data was taken from a sedan implemented in VDL. The majority of the thermal parameters were estimated from authors' experience with similar systems in industrial applications.

### 2.1 System Model

Figure 1 shows the integrated VTM model. This model is structured in a thermal-centric way such that the key thermal subsystems are visible at the top level of the model and in parallel with the vehicle model. At this level, the model contains the following subcomponents with key systems to be described in more detail in subsequent sections:

- Lumped 1D conventional vehicle model with automatic transmission
- Lumped thermal models for engine and transmission
- Simple underhood models for engine and transmission heat transfer
- Controllers for fan and grill shutters
- Coolant and transmission oil fluid circuits
- Heat exchanger stack with radiator, condenser, transmission oil cooler, and charge air cooler
- Minimal HVAC and charge air circuits
- Electric fan with simplified vehicle electrical system

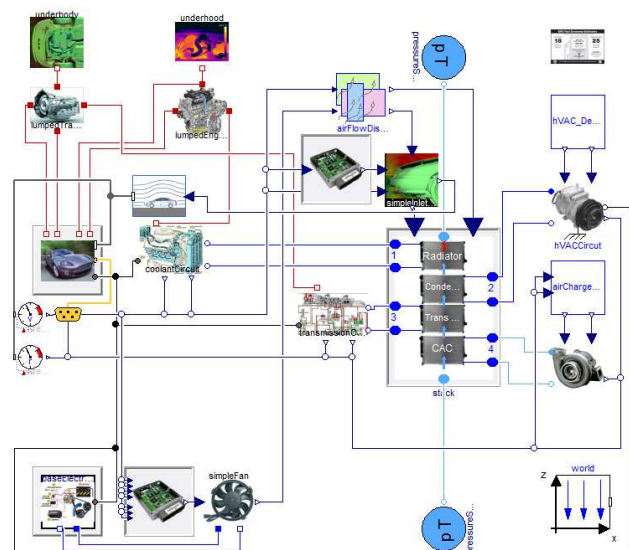


Figure 1. Integrated VTM model

## 2.2 Vehicle Model

The vehicle model is implemented using the vehicle model architecture and components from Vehicle Dynamics Library [3]. This architecture supports the full range of vehicle models from lumped 1D to geometry-based full 3D representations suitable for vehicle dynamics and handling applications. Previous work [9] has illustrated drivability applications using Vehicle Dynamics Library with simplified chassis representations. A similar approach is used to create a computationally-efficient lumped 1D vehicle model suitable for drive cycle simulations. This model focuses on core loads and losses to ensure that the engine operates at the appropriate operating conditions for accurate fuel consumption and heat generation. Both the mapped engine and transmission generate heat input to the thermal system via thermal connectors added to the vehicle architecture. The simplified chassis model with rigid axle, no slip tires, and a single lumped mass is shown in Figure 3. Parameterization for the vehicle model is taken from a sedan model in VDL.

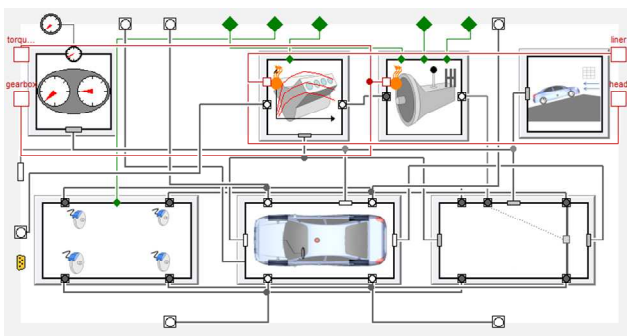


Figure 2. Vehicle model augmented with thermal behavior

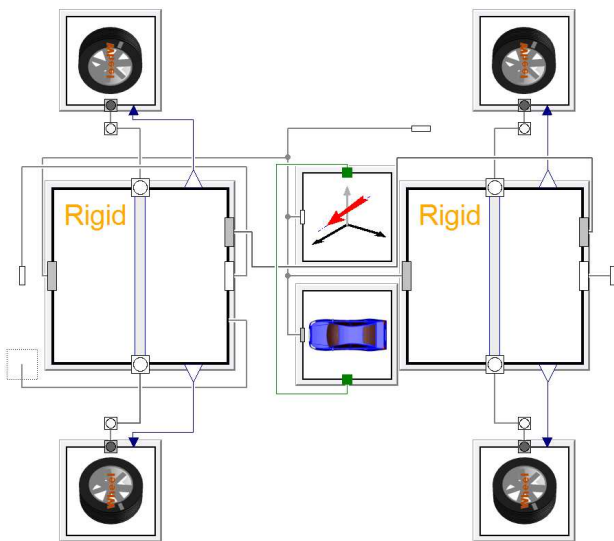


Figure 3. Simplified 1D chassis model

## 2.3 Fluid Circuits

Fluid circuit modeling is a key part of the VTM model. An efficient thermo-fluid implementation is critical as it is common for a VTM model to include several different fluid circuits which interact with the thermal system and heat exchanger stack. Liquid Cooling Library [4] is ideal for modeling incompressible fluid circuits due to its efficient formulations which can support models with minimal number of pressure states, potentially even a single pressure state per circuit, while maintaining thermal states as required throughout the system. This approach eliminates the stiffness and resulting computational impact of modeling incompressible flow circuits as minimally compressible.

Figure 4 shows a simple coolant circuit implemented to support the demonstrator VTM model. This model includes the following components from Liquid Cooling Library:

- Coolant pump driven by crankshaft
- Coolant path through engine head and block
- Thermostat to control flow to radiator and via bypass
- Expansion volume

A similar model is implemented for the transmission oil circuit.

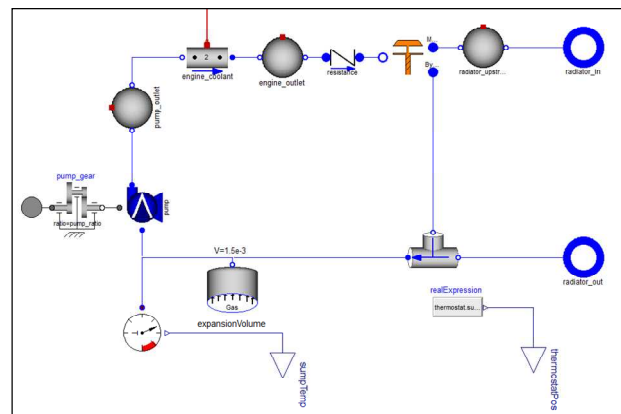


Figure 4. Simple coolant circuit

## 2.4 Heat Exchanger Stack

The heat exchanger stack is a key coupling point between the fluid circuits. Both Liquid Cooling Library and Heat Exchanger Library include models of heat exchanger stacks. These models differ in terms of model detail. Heat Exchanger Library allows detailed, geometry-based models which can be stacked using a streamtube approach for the airflow that preserves non-uniform conditions for each heat exchanger throughout the stack. The heat exchanger models in Heat Exchanger Library can be discretized

and can accept both uniform and non-uniform air-flow inputs. The approach and a sample model are shown in Figure 5. This formulation is efficient enough to support drive cycle work assuming a reasonable number of stream tubes. The stack models in Liquid Cooling Library require only the basic stack geometry and can use mapped heat exchangers with imposed airflow based on external inputs from CFD, etc. The air temperatures through the stack are calculated based on the stack geometry with a single calculated input temperature for each heat exchanger based on preceding heat exchanger outlet temperatures. LCL includes assembled stack models ranging from 2-8 heat exchangers.

Figure 6 shows the heat exchanger stack used in the VTM model. The stack consists of a radiator, condenser, transmission oil cooler, and charge air cooler. Figure 6 shows the visualization of the stack geometry which also provides dynamic visualization of the temperatures during the run. The dynamic summary visualization for the temperatures, flowrates, and heat transfer in each heat exchanger is also shown. Each heat exchanger is implemented as a mapped effectiveness as a table of the fluid and air mass flow rates. Though simplified, effectiveness data is often available early in vehicle programs and thus can support upfront cooling pack concept assessment and early thermal system design and performance.

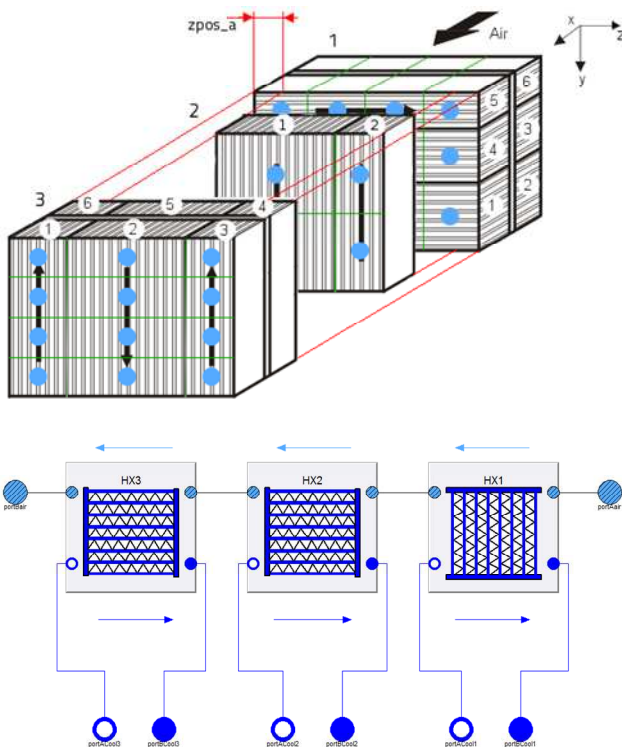


Figure 5. Streamtube approach and sample stack from Heat Exchanger Library

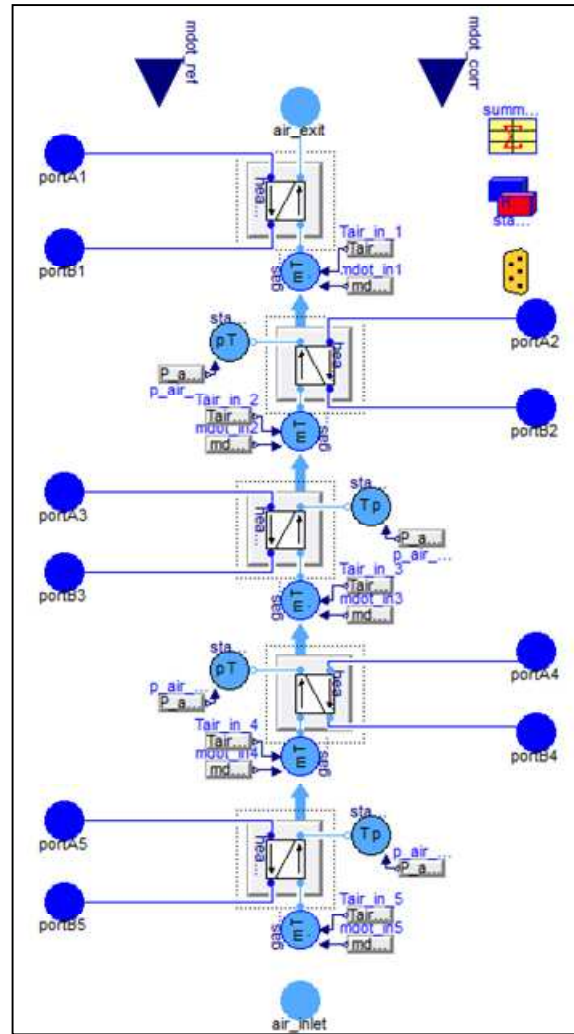


Figure 6. Heat exchanger stack

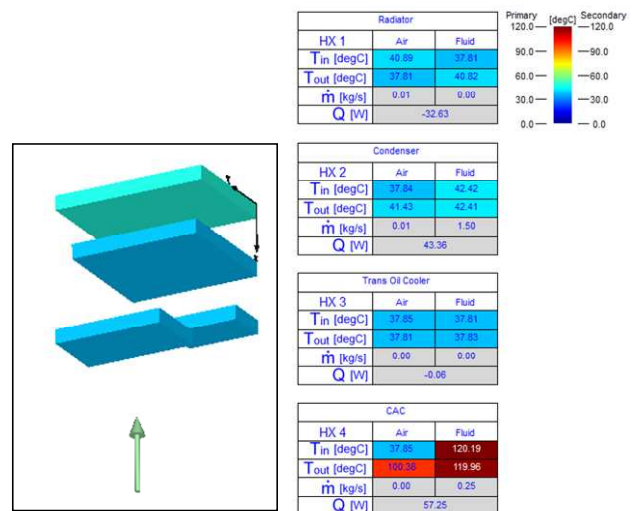


Figure 7. Stack geometry and summary visualization

## 2.5 Controllers

For demonstration purposes, simple controllers in Modelica are implemented for the fan and grill shut-

ters. The fan controller is shown in Figure 8. The grill shutter implementation mimics a passive grill shutter system where grill shutters are closed above a parameterized vehicle speed as shown in Figure 9. More detailed controller implementations in Simulink are discussed in Section 3.2 when the VTM model is exported as an FMU and combined with the control system in Simulink.

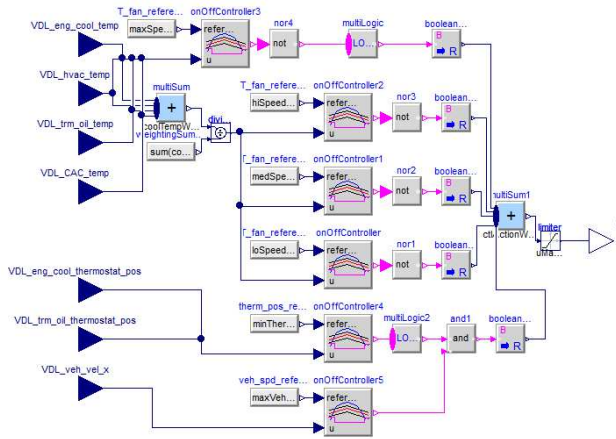


Figure 8. Simple fan controller

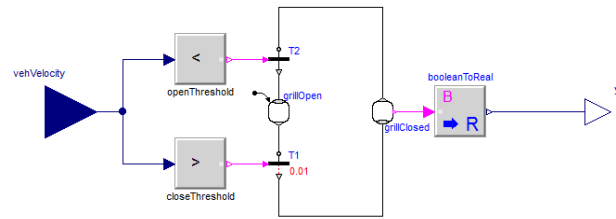


Figure 9. Passive grill shutter control

### 2.6 Driver and Drive Cycles

To support drive cycle simulations, the vehicle model shown in Figure 2 is augmented with a driver model as shown in Figure 10. The driver model provides closed loop trace following based on the vehicle longitudinal velocity. The driver model also handles the gear shifting for the automatic transmission based on a shift map. The driver model is adapted from the closed loop driver in VDL. Ambient and road conditions are also specified in the augmented driver.

Drive cycle selection is also implemented in conjunction with the vehicle model. Figure 11 shows the interface for drive cycle selection. Common and publicly available drive cycles are selectable from a drop down list. Custom drive cycles are supported via the “User Defined” option where the cycle data is implemented directly in the model or via the “File” option where the drive cycle data is read from a file.

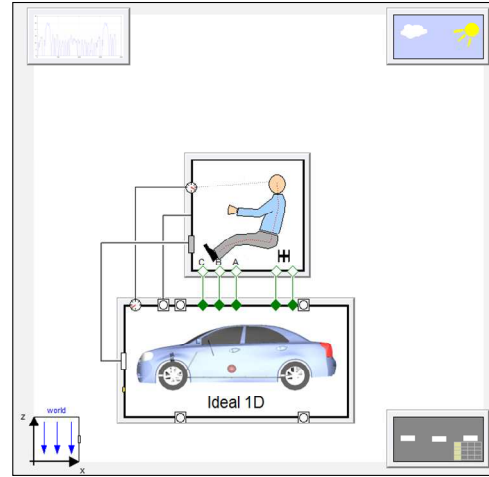


Figure 10. Assembled vehicle with driver, drive cycle, and ambient models

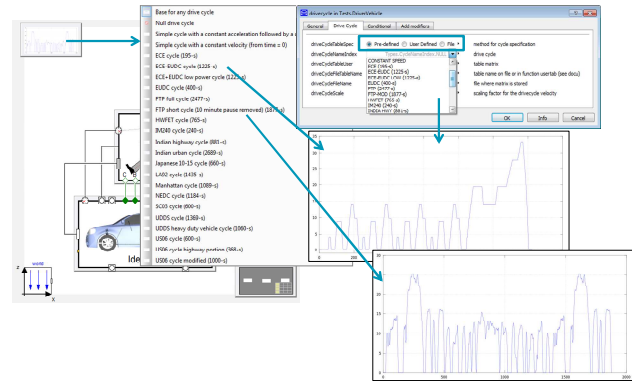


Figure 11. Drive cycle model with standard and user-defined cycle implementations

## 3 Application Examples

Using the component and subsystem models outlined in the previous section, this section details several applications and workflows using the integrated VTM model.

### 3.1 Drive Cycle Simulation

The first application simply illustrates drive cycle simulations in Dymola [10] with the models described in Section 2. The integrated VTM model is simulated on the US06 drive cycle. Selected results from that simulation are shown in Figure 12. These results illustrate both the typical drive cycle simulation results along with results of a thermal system such as coolant and oil temperatures. Even with parameterization for a demonstrator model, the results are certainly reasonable and suitable to illustrate additional workflows with the VTM model.

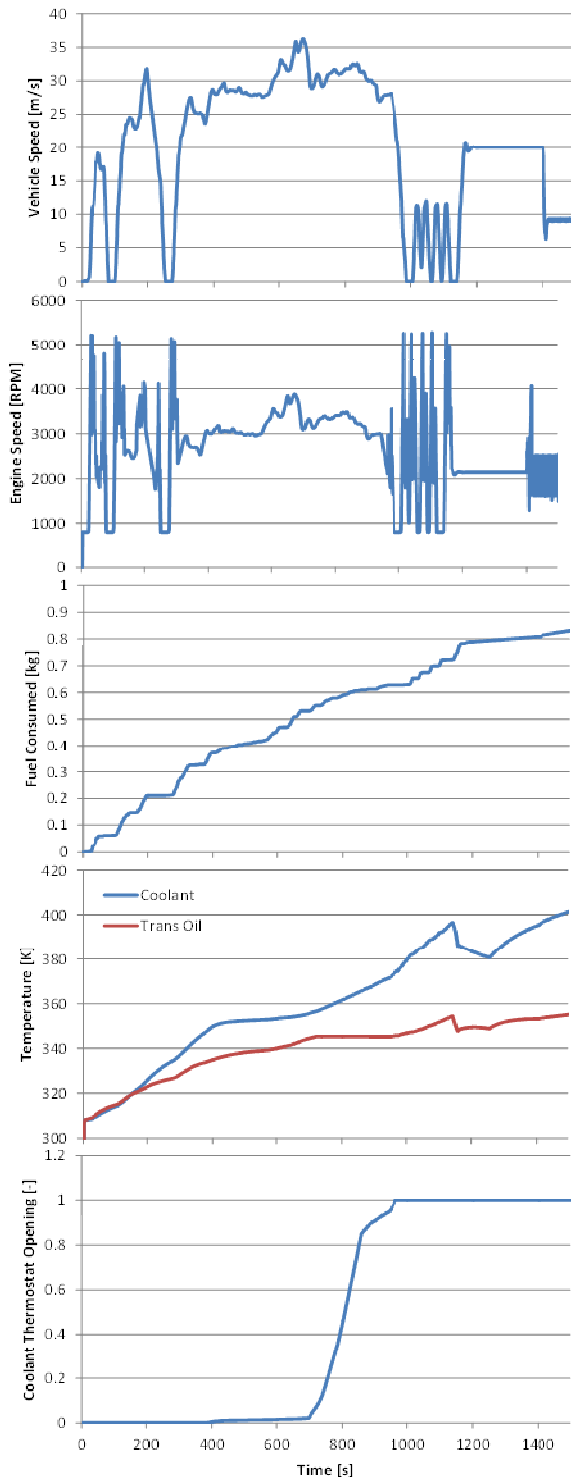


Figure 12. Simulation results on US06 drive cycle

### 3.2 Controls Development and Optimization

While Modelica can handle both controls and physical models, a common workflow is to combine Modelica-based tools for physical modeling with Simulink for controls development. This coupling between Simulink and Modelica-based tools like Dymola is especially streamlined with Functional Mock-up Interface (FMI) [6]. While Simulink is not

natively FMI-compliant, Modelon provides the FMI Toolbox for MATLAB [7] that enables both import of FMUs into Simulink and export of FMUs from Simulink. This bi-directional coupling is extremely powerful and useful in that it allows both controls engineers and physical system modelers to leverage best-of-breed tools to support their work with a robust and straight-forward workflow for integration in either simulation environment.

For integration with controls in Simulink, the integrated VTM model in Figure 1 is simply modified to accept external inputs for the fan and grill commands as shown in Figure 13 to provide a partitioning between the physical and control systems. A model exchange FMU is created in Dymola from this model. This FMU for the VTM model is then imported into Simulink using FMI Toolbox for MATLAB and integrated with the fan and grill controllers implemented natively in Simulink. The resulting integrated model in Simulink is shown in Figure 14.

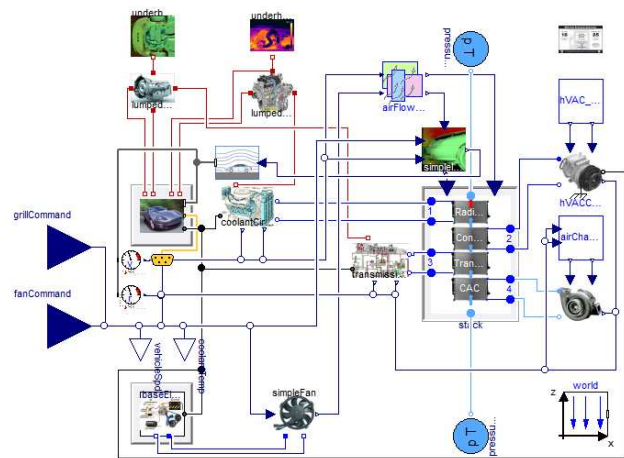


Figure 13. VTM model with external control for FMU generation

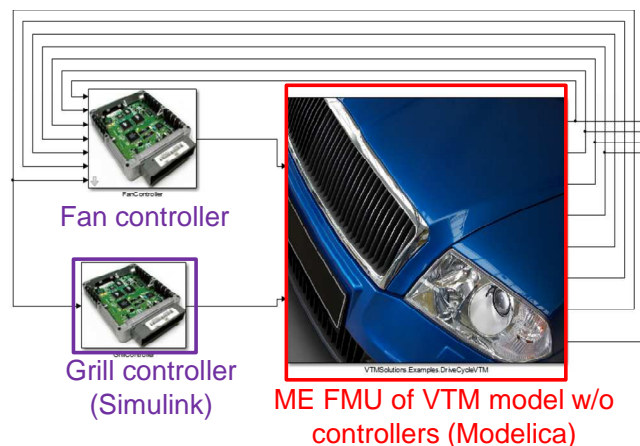


Figure 14. Integrated model in Simulink via FMU import with FMI Toolbox for MATLAB

An active shutter control strategy was developed in Simulink to explore opportunities for improved vehicle fuel consumption. To help determine optimal settings for grill command and fan command, the DOE capabilities of FMI Toolbox for MATLAB to execute the runs and post-process results were used to run a full factorial sweep for grill and fan command at different vehicle speeds. Sample results for a vehicle speed of 120kph are shown in Figure 15. These DOE results can be used to identify optimal grill and fan commands that maintain desired coolant temperatures at minimum fuel consumption.

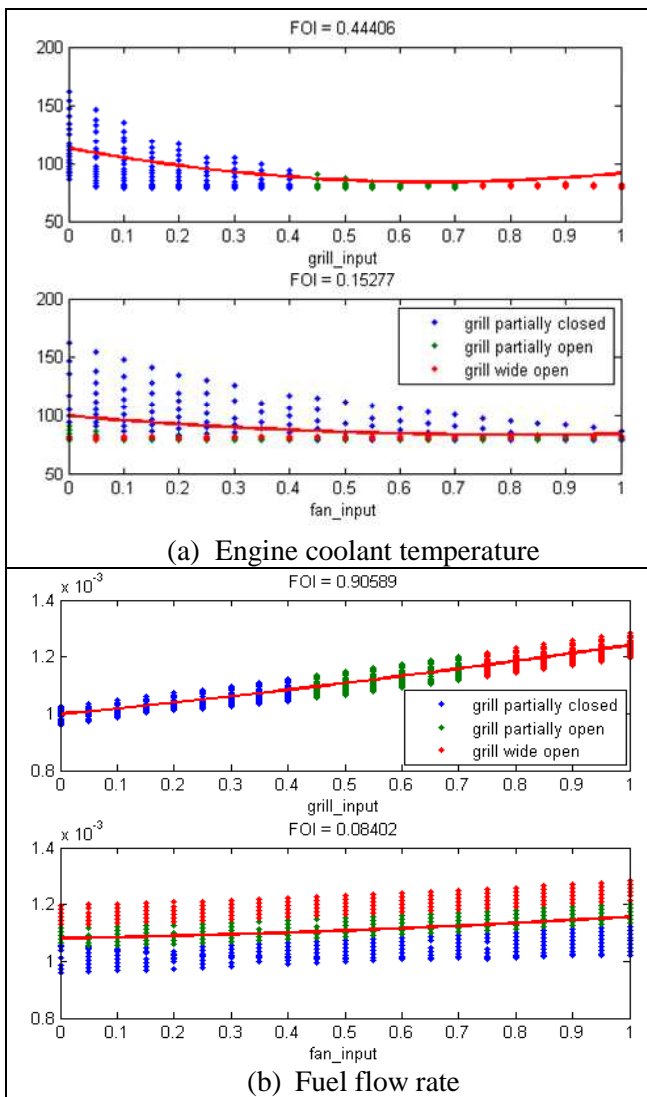


Figure 15. Results from DOE for grill and fan commands at 120kph vehicle speed

Using the DOE results, an active control strategy is implemented in Simulink. The implementation is shown in Figure 16. This strategy attempts to keep the grill closed as much as possible to reduce aerodynamic drag while maintaining target coolant and oil temperatures. The controller attempts to use op-

timal grill and fan settings when operating in cooling mode.

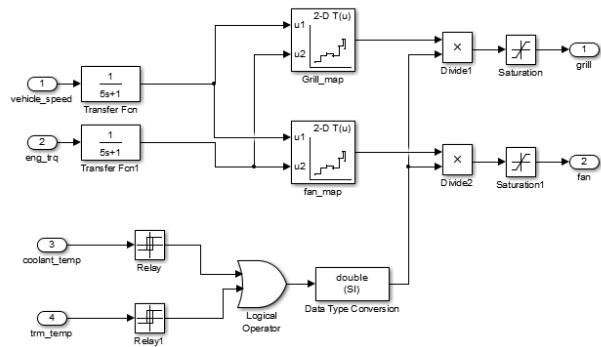


Figure 16. Active grill shutter controller

The active and passive strategies were run on a number of common drive cycles. Figure 17 shows the vehicle speed for the FTP MOD drive cycle. Figure 18 shows comparisons between the active and passive grill shutters for the FTP MOD drive cycle. For this drive cycle, the vehicle speeds are less than 15 m/s for the majority of the cycle. The coolant temperatures between the mechanical and active shutter controllers are compared in Figure 18. The active shutter controller can be observed to speed the warm up of the coolant temperatures by at least 2 minutes in addition to maintaining the coolant temperatures close to the desired operating point of 90°C throughout the entire drive cycle. The grill and fan commands in the mechanical shutter controller switch between off and completely on whereas the active shutter controller makes optimal use of the cooling mechanism to both speed up the warming process and at the same time maintaining temperatures close to the desired operating point. When compared at equivalent coolant temperatures, the optimal controller is expected to have slightly better fuel consumption (typical benefits of grill shutters are in the range of 0.5-2% depending on the vehicle and cycle). Of course, these results are highly dependent on the power consumption of the fan and the tradeoff between fan power and recovered aero drag and highlight the need for an analytic model to comprehend these complex tradeoffs.

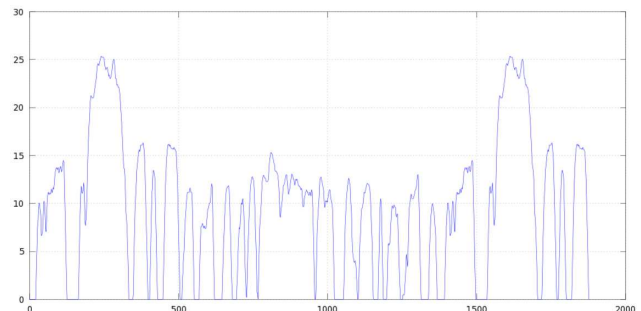


Figure 17. Vehicle speed for the FTP MOD drive cycle

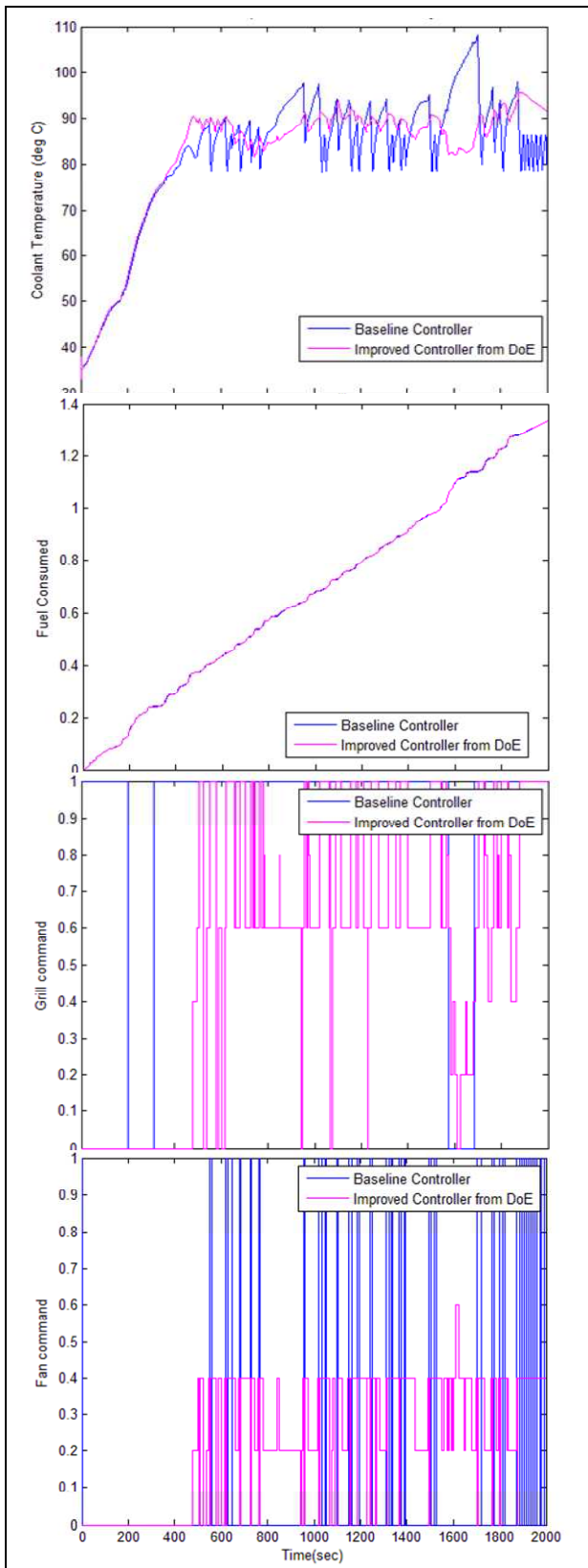


Figure 18. Comparisons between active and passive grill shutters on FTP MOD cycle

### 3.3 Batch Simulations and Robustness

The integrated VTM model can be simulated from Excel using FMI Add-in for Excel to support batch

simulations and robustness applications. To support this workflow, a co-simulation FMU is created and then imported into FMI Add-in for Excel. Experiment sheets allow users to change parameters, provide external inputs, apply boundary conditions, simulate the model, and post-process the results. The simulations are automatically run in parallel distributed across the local CPU cores. The use of FMI for model deployment outside of the model development environment is providing additional value from a standards-based workflow.

A sample experiment sheet for the VTM model is shown in Figure 19 to run the model over various drive cycles. Coolant temperature results from a batch simulation at a range of vehicle speeds are shown in Figure 20. Using the scripting API provided with FMI Add-in for Excel, Monte Carlo simulations for robustness applications are enabled. Figure 21 shows the Monte Carlo experiment sheet created by the script and results from simulations over a distribution of heat exchanger effectiveness multipliers and airflow distribution multipliers.

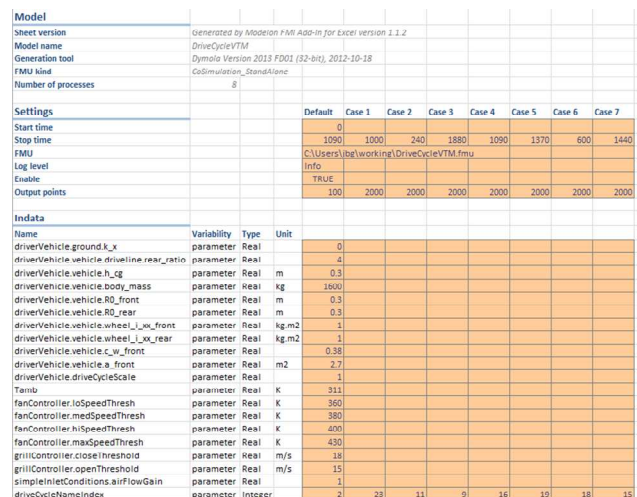


Figure 19. Experiment sheet in Excel to run model over different drive cycles

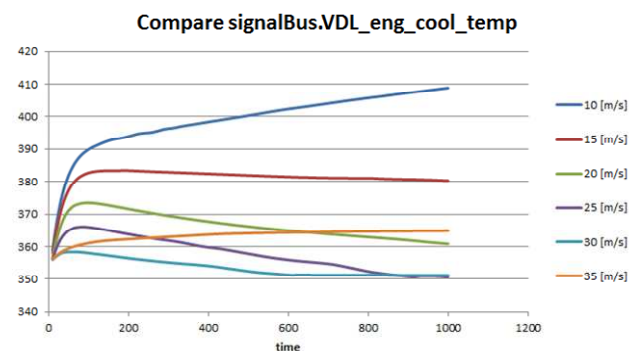


Figure 20. Batch simulation showing coolant temperature over a range of vehicle speeds



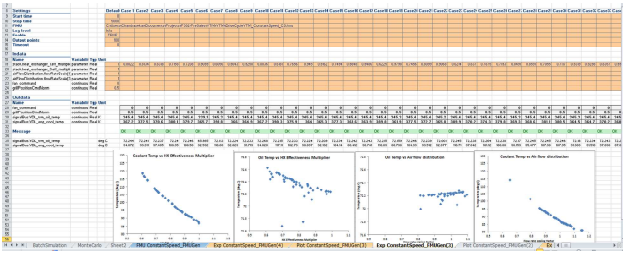


Figure 21. Monte Carlo simulations for heat exchanger effectiveness and stack airflow

### 3.4 Heat Exchanger Non-Uniformity

Heat exchanger performance is a critical factor in VTM applications. While convenient to simulate with uniform velocity and temperature inputs, actual conditions typically include non-uniformity. Thus, assessing heat exchanger performance under non-uniform conditions is critical. Heat Exchanger Library [5] provides both uniform and non-uniform input sources. Figure 22 shows a test model that can be configured for either uniform or non-uniform inputs. Non-uniform inputs would typically be provided from a CFD tool and thus represent a 1D-3D coupling between CFD and the discretized 1D approach in the models in Heat Exchanger Library. Similar coupling with the Air Conditioning Library [12] for evaluating idle air recirculation has been published [13].

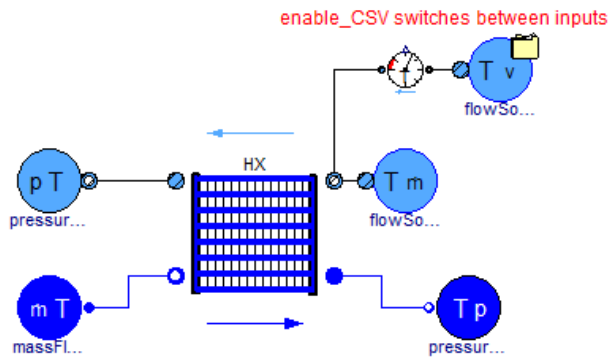


Figure 22. Heat exchanger test with option for uniform or non-uniform inputs

To verify the model, published distributions [11] were simulated and compared with analytic results from the publication for heat exchanger non-uniformity, defined as the ratio of heat transfer for non-uniform inputs to heat transfer with averaged uniform inputs from the non-uniform distribution:

$$NonUniformityFactor = \frac{Q_{non-uniform}}{Q_{uniform\ avg}} \quad (1)$$

The study was performed with a model from Heat Exchanger Library calibrated to bench data. The

model was then run over the published distributions in [11] for a range of heat capacity ratios with the external air as the minimum heat capacity fluid and non-uniformity results compared with published values. Figure 23 shows a sample distribution. The comparisons between the HXL simulation and the published results are shown in Table 1. Note that the results from the paper were extracted from graphs in [11]. The simulations in [11] were also run over a large range of NTU values (0-100) and it was difficult to extract values at the NTU for this cooler (roughly 1-2). Thus, the values for NTU=5 and NTU approaching zero were extracted for comparison with the model and shown in the table. The model accurately captures both the trend and magnitude of the non-uniformity.

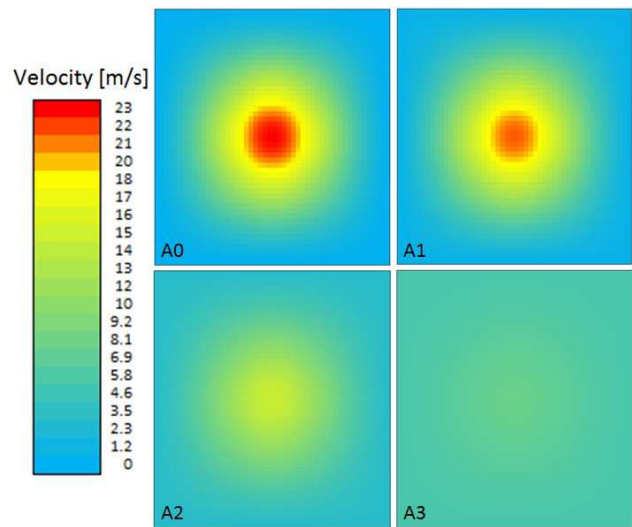


Figure 23. Velocity distributions from patterns in [11]

Table 1. Non-uniformity comparison between model and published results [11]

Distribution	Cmin/Cmax	HXL Simulation	Ranganayakulu Paper	
		Non-Uniformity Interpolated	NTU = 0 Non-Uniformity	NTU = 5 Non-Uniformity
A0	0.2	0.820	0.818	0.846
A1	0.2	0.859	0.856	0.874
A2	0.2	0.959	0.950	0.958
A3	0.2	0.998	0.991	0.993
A0	0.4	0.815	0.769	0.850
A1	0.4	0.860	0.831	0.900
A2	0.4	0.961	0.948	0.956
A3	0.4	0.998	0.993	0.995
A0	0.6	0.814	0.750	0.821
A1	0.6	0.862	0.810	0.862
A2	0.6	0.962	0.946	0.949
A3	0.6	0.998	0.991	0.991
A0	0.8	0.815	0.811	0.809
A1	0.8	0.864	0.844	0.840
A2	0.8	0.963	0.946	0.938
A3	0.8	0.998	0.989	0.989
A0	1	0.816	0.810	0.808
A1	1	0.866	0.845	0.842
A2	1	0.964	0.971	0.945
A3	1	0.998	0.999	0.998

## 4 Conclusions

A coordinated suite of Modelica libraries for vehicle thermal management applications have been used in the implementation of an integrated VTM model with combined vehicle fuel and thermal effects, including the key physical and control models. The demonstrator VTM model is implemented using the Vehicle Dynamics Library, Liquid Cooling Library, and Heat Exchanger Library from Modelon. Several application examples focused on vehicle thermal management have been detailed. These application examples include drive cycle simulations, controller development and optimization, batch simulations and robustness applications, and 1D-3D coupling for heat exchanger performance. These application examples demonstrate the use of sophisticated model libraries to enable the multi-domain approach needed for vehicle thermal management applications. The application examples also illustrate the use of FMI to couple the VTM model with controls in Simulink and for use in robustness application in Excel. Given the importance of model deployment outside of traditional CAE environments to support model-based systems engineering, workflow aspects via FMI are highlighted.

## References

- [1] Bouvy, et al., “Holistic Vehicle Simulation using Modelica – An Application on Thermal Management and Operation Strategy for Electrified Vehicles”, *Proceedings of 9<sup>th</sup> International Modelica Conference*, pp. 263-270, 2012.
- [2] King, Jenny., “Shooting the breeze about active grille shutters: What do they do?”, <http://cars.chicagotribune.com/fuel-efficient/news/chi-active-grille-shutters-20130613>, June 13, 2013.
- [3] Andreasson, J., “The Vehicle Dynamics Library: New Concepts and New Fields of Application”, *Proceedings of 8<sup>th</sup> International Modelica Conference*, 2011.
- [4] Modelon, “Liquid Cooling Library”, Version 1.1, 2013. <http://www.modelon.com/products/modelica-libraries/liquid-cooling-library/>
- [5] Modelon, “Heat Exchanger Library”, Version 1.0, 2013. <http://www.modelon.com/products/modelica-libraries/heat-exchanger-library/>
- [6] MODELISAR, “Functional Mock-up Interface for Model Exchange”, Version 1.0, 2010.
- [7] Modelon, “FMI Toolbox for MATLAB”, Version 1.7, 2013. <http://www.modelon.com/products/fmi-toolbox-for-matlab/>.
- [8] Modelon, “FMI Add-in for Excel”, Version 1.2.1, 2013. <http://www.modelon.com/products/fmi-add-in-for-excel/>
- [9] Griffin, J., Batteh, J., and Andreasson, J., “Modeling Vehicle Drivability with Modelica and the Vehicle Dynamics Library”, *Proceedings of 9<sup>th</sup> International Modelica Conference*, pp. 599-608, 2012.
- [10] Dassault Systemes, “Dymola 2013 FD01”, 2012.
- [11] Ranganayakulu, CH., Seetharamu, K.N., and Sreevatsan, K.V., “The Effects of Inlet Fluid Flow Nonuniformity on Thermal Performance and Pressure Drops in Crossflow Plate-Fin Compact Heat Exchangers”, *International Journal of Heat and Mass Transfer*, Vo. 40, No. 1, pp. 27-38, 1997.
- [12] Eborn, et al., “AirConditioning - a Modelica Library for Dynamic Simulation of AC Systems”, *Proceedings of 4<sup>th</sup> International Modelica Conference*, pp. 185-192, 2005.
- [13] Wang, et al., “Integrated Thermal Management Simulations: Evaluating the Effect of Underhood Recirculating Airflows on AC-System Performance”, *Proceedings of 7<sup>th</sup> International Modelica Conference*, pp. 413-422, 2009.

# Virtual Integration for hybrid powertrain development, using FMI and Modelica models

Lionel Belmon<sup>1</sup>

Yujung Geng<sup>1</sup>

Huaqiang He<sup>2</sup>

<sup>1</sup>: Global Crown Technology, Beijing, China  
[Lionel.Belmon@globalcrown.com.cn](mailto:Lionel.Belmon@globalcrown.com.cn)

<sup>2</sup>: Dongfeng Commercial Vehicles Technical center, Wuhan, China  
[hehuaqiang@dfcv.com.cn](mailto:hehuaqiang@dfcv.com.cn)

## Abstract

Dongfeng Commercial Vehicles (DFCV) is developing powertrain controls for a hybrid light truck. To support this development, a virtual integration platform is being implemented, using Modelica models and Functional Mock-up Units (FMUs) for the engine/EMS, gearbox, MCU/e-motors, driveline, tyres and longitudinal dynamics. Simulink models and/or c-code of the Hybrid Control Unit (HCU) and Transmission Control Unit (TCU) are also integrated in the platform to achieve closed-loop simulation. The virtual integration allows reproducing accurately the overall vehicle behavior and is used for optimization of gearshifts, hybrid mode switches and hybrid drive strategies.

*Keywords: Hybrid powertrain, FMI, Control software*

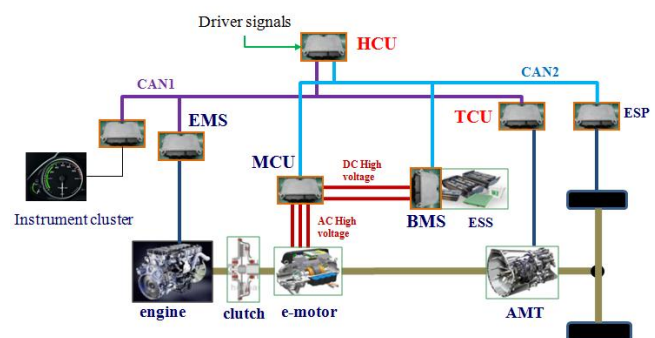
## 1 Motivation and objectives

The control systems of hybrid powertrain are generally implemented using several ECUs networked together. Functions are distributed through the controllers. Typically, the powertrain controllers will include a Hybrid Control Unit (HCU), an Engine Management System (EMS), a Transmission Control Unit (TCU), a Motor Control Unit (MCU), and a Battery Management System (BMS). Other controllers from chassis systems (ESP/ABS) might also interact with the powertrain.

The development of such a system requires considering the interactions of all main components together, namely engine, gearbox, actuation system, e-motors,

battery, driveline, tyres. For the control software, it also implies that functions cannot be developed independently but are now inter-dependent and distributed over several ECUs. This poses challenges to classical development processes.

Dongfeng Commercial Vehicles is developing such a hybrid powertrain for application in a light truck. We give a schematic of the system under consideration in Figure 1. DFCV wishes to have an efficient tool for integrating HCU and TCU control logic, optimizing parameters and performing system testing. This tool should be available for function developers, should be cost-efficient and deployable.



**Figure 1 : schematic of the hybrid powertrain**

The objective of the project described in this paper is thus to establish a *virtual hybrid powertrain* where subsystems and controllers can be simulated together on a standard PC. Function developer can then easily verify on their PC the behavior of any changes in software or parameters. This virtual powertrain can also be used for other applications, such as large coverage testing or system/parameters optimization.

For creating this *virtual hybrid powertrain*, the following tools are used:

### **ITI SimulationX – plant models authoring**

SimulationX is a Modelica platform that provides powerful commercial libraries for powertrain modeling and electrical systems modeling [1]. The tool fully support the FMI standard and application has been demonstrated for powertrain applications [2]. Models can be exported to a standard format and can be executed without license restrictions. This is an important feature for deployment of the *virtual powertrain*.

### **Simulink – HCU and TCU software authoring**

Dongfeng Commercial Vehicles developed HCU and TCU control software using a model-based approach in Simulink. Simulink floating-point models can be compiled and integrated in the virtual powertrain. This would be then a “model-in-the-loop” setup. It is also possible to use the final production-code (fixed point) and calibration parameters for integration inside the virtual powertrain. This method could be described then as Virtual ECUs. [3]

### **QTronic Silver – Integration platform**

QTronic Silver is an integration platform, widely used for powertrain applications [4],[5],[6]. Silver provides the simulation core, an interactive GUI dashboard, numerous interfaces and tools for integrating plant models and control software or even performing ECU chip simulation. The Virtual ECUs described in the previous paragraph are built with the Silver Basic Software (SBS) technology.

### **QTronic TestWeaver – Large coverage testing**

Powertrain systems, and in particular hybrid powertrain systems are systems difficult to test because of the very large number of system states and larger number of state transitions. For instance, a relevant test campaign should test all gearshifts, in various slopes. It should also test all hybrid mode transitions, under various State Of Charge (SOC). The test space is huge and test scripts/manual testing is not an efficient method. QTronic TestWeaver is an intelligent test system that can generate test cases to increase test coverage, drive the system under test to uncovered states and report problems/bugs when these are met. TestWeaver has been successfully applied in a large number of powertrain projects [5],[7],[8].

## **2 Plant models**

The plant models to be developed will be used for development support of HCU and TCU. This means that all subsystems and all remaining controllers will be modeled in the plant. This includes in particular control models for EMS, MCU and BMS.

We can define several general requirements for the plant models :

- Simulate all required bus&sensors signals (>200)
- Simulate EMS,MCU and BMS logical functions
- Simulate the necessary physics
- Simulate fast enough for convenient use
- Simulation should be accurate enough to support optimization of relevant system parameters

We describe now how plant models are developed.

### **2.1 CAN buses**

The hybrid powertrain under consideration use 2 CAN networks, CAN1 and CAN2. Dongfeng Commercial Vehicle provided the complete list of CAN messages. The 2 CAN networks are implemented as Modelica connectors.

The connectors are then used in the simulated controllers for EMS, MCU and BMS. They are also used for defining I/O variables of the FMU that will be exported from the vehicle model. The CAN1/CAN2 networks have together around 200 signals. In SimulationX, the network is considered as ideal without losses or delays besides the ECUs task cycle time. Non-ideal behavior of the CAN network is implemented and simulated in QTronic Silver which supports special features for this.

### **2.2 Automated Manual Transmission gearbox**

The AMT model consists of input and output inertias, gear stages and synchronizers. Drag torques and efficiency losses are also included. The AMT stiffness is lumped inside the synchronizers hubs.

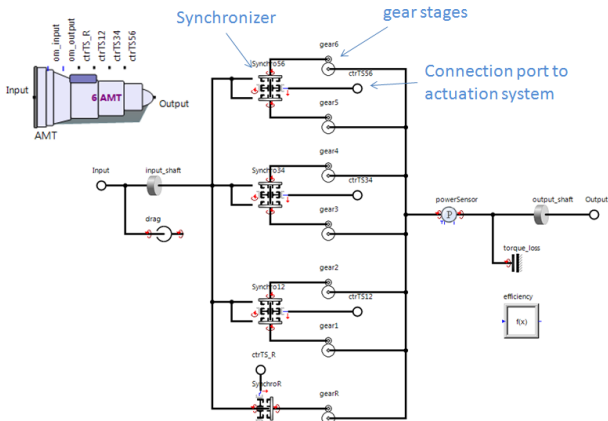


Figure 2 : AMT gearbox model

The synchronizer models are complex behavioral models that fully reproduce the synchronization process, with synchronization torque depending on actuator force. The synchronizer behavioral model has been compared to full contact-based synchronizers models in [9]. Some results of the comparisons are given in Figure 3. The detailed synchronizer model with dog-clutch gear contacts has been validated in [10].

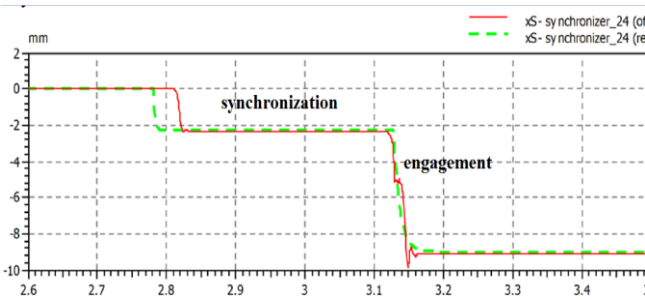


Figure 3 : Comparison of synchronization and engagement process for detailed (red)/functional (green) synchronizer models

We can conclude that the behavioral model computes correctly the synchronization time and the engagement process. The behavioral model has the advantage of requiring few parameters and of being a “fast-running” model. It has been applied in the past in real-time simulations for HiL [9]. The synchronization time is an important quantity for the gearshifting simulation and must be simulated correctly.

### 2.3 Clutch model

The clutch model can be separated in 3 sub-components:

#### Friction/torque model:

The torque capacity of the clutch is defined as a function of the clutch actuator position. This is implemented through a look-up table. Clutch wear is

included in the modeling. Finally this torque capacity is used in a stick/slip friction model.

#### Spring force on actuator side:

The clutch actuator has to overcome the conical spring force. The spring is non-linear and has a strong hysteresis. This is implemented using a hysteresis table from SimulationX

#### Thermal model:

The clutch friction surfaces temperature is an important quantity to simulate. This is done by computing losses in the clutch during slip and using thermal capacities and heat transfer models.

### 2.4 Gearbox actuators

The gearbox actuation system is a pneumatic actuation system controlled by solenoid valves. The model is created using the SimulationX pneumatic library, where pressure/temperature are computed using mass/energy balances and compressible flow equations. The solenoid valves receive PWM signals from the TCU. The pneumatic actuation model includes all solenoid valves, gearshift cylinders and clutch cylinders. A selector gate model is also implemented. We give the overview of the pneumatic actuation model in Figure 4

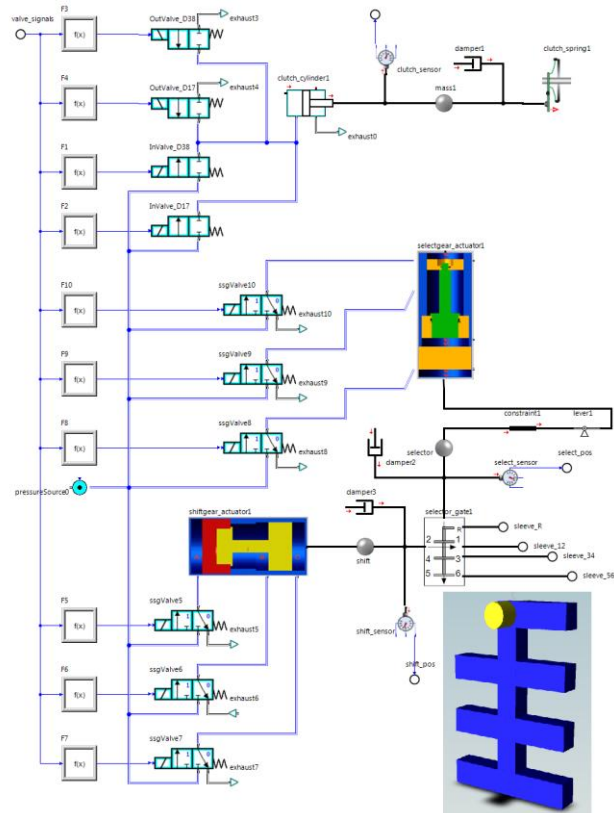
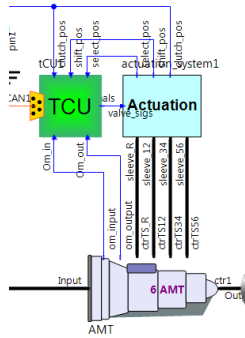


Figure 4 : Pneumatic actuation model – integrated in the vehicle model

The pneumatic actuation model can reproduce the clutch actuation dynamics. In particular, the clutch

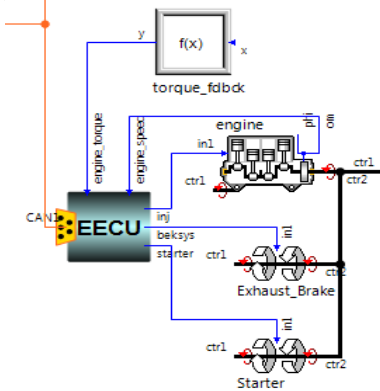
actuation is a closed control with a position feedback. The model can be used for tuning of control gains and control laws.



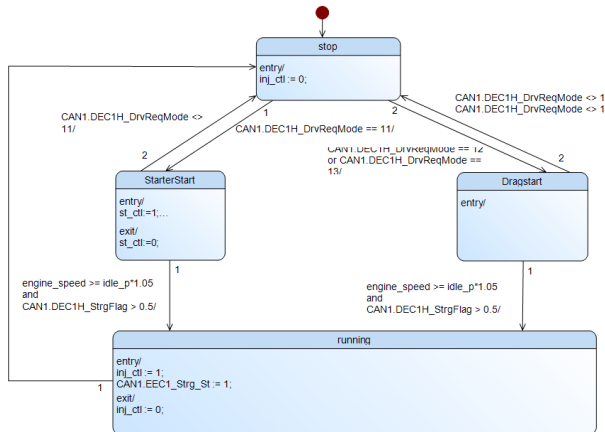
**Figure 5 : AMT with “packaged” actuation system and a soft-TCU model**

**2.5 Engine and EMS**

The engine and EMS models are based on tables defining the torque/rpm capacity of the engine, along with the fuel consumption. Turbocharger dynamics are so far neglected, but the turbocharger delay in boost pressure will probably be implemented in a model revision. Besides this rather simple table-based modeling, a state-chart is also added for simulating the logical behavior of the EMS.



**Figure 6 : Engine and EMS model**

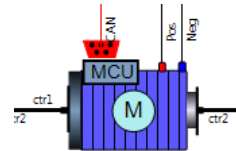


**Figure 7 : EMS state-chart**

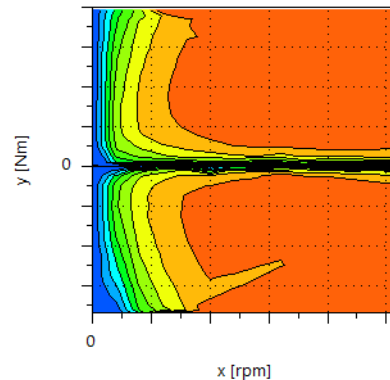
The EMS model must simulate the various operation mode of the engine, according to CAN requests. For instance, the EMS have a self-start mode and a “drag start” mode where the e-motor will start the engine by closing the clutch. The state-chart editor of SimulationX is thus very valuable for creating such models.

**2.6 E-motor and MCU**

The e-motor model is based on an energy/power approach. The AC 3-phase modeling is not considered and we only focus on the DC interface. The model is thus based on an efficiency approach in which the current on the DC side is computed as a function of DC voltage and motor torque. As for the EMS, a state-chart is also included in the MCU model to represent the logical behavior of the MCU software.



**Figure 8 : e-motor and MCU model**



**Figure 9 : efficiency map in the e-motor**

Motor cooling circuit is so far not modeled in detail because the focus is on HCU and TCU.

**2.7 Battery and BMS**

The battery model is based on a Open Circuit Voltage table and on parasitic and polarization resistance/capacity. The diagram of the model is shown below.

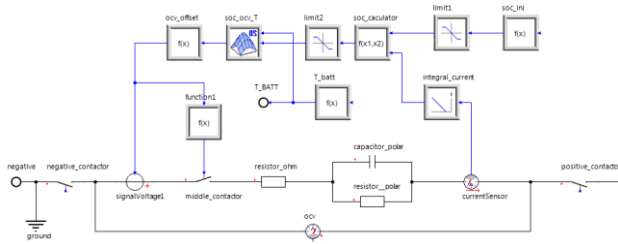


Figure 10 : Diagram of the Battery model

As for the EMS and MCU, the Battery Management System (BMS) uses a state-chart for handling the various states and transitions of the BMS software. The model is finally packaged as shown in Figure 11.

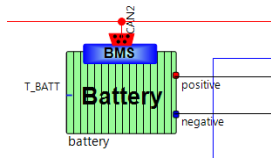


Figure 11 : Packaged battery and BMS model

### 2.8 Vehicle model overview

We give an overview of the vehicle model in Figure 12.

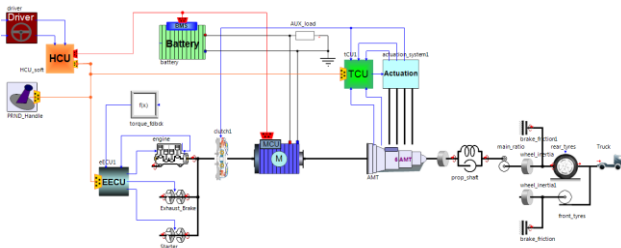


Figure 12 : Diagram view of vehicle model

The vehicle model includes tyre with slip and driveline/propeller shaft with stiffness. The stiffness of the driveline is so far lumped and set according to an equivalent stiffness computed from CAD drawings. This assumption will probably be reviewed in future model updates.

### 2.9 Export of the model as an FMU

The plant model input/outputs are defined in the Code Export wizard of SimulationX. In particular the definition of CAN buses is helpful. The model is exported so that exported FMU variable names correspond exactly to the names defined in the CAN bus definition. This will be an important property when doing integration in Silver.

## 3 Hybrid powertrain integration

The integration flow of the HCU/TCU control software and plant is summarized in the figure below. The plant model is exported from SimulationX as a FMU, using the Functional Mock-up Interface. The control models are built using Simulink Coder and Silver Basic Software scripts.

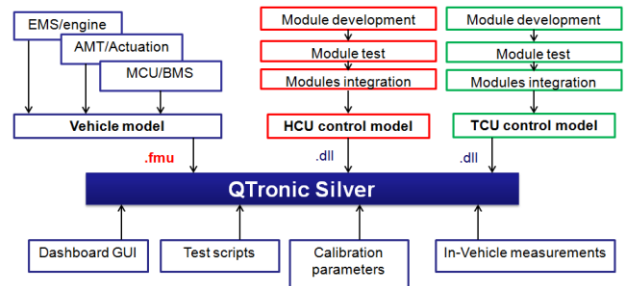


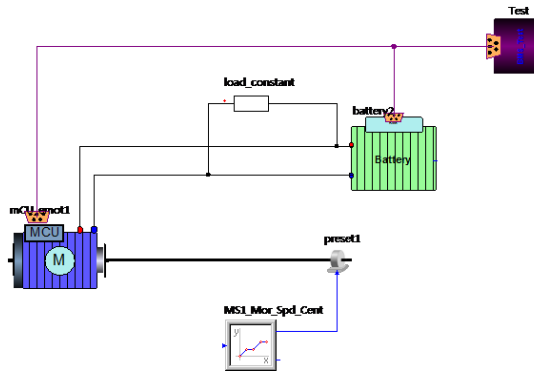
Figure 13 : Integration flow for plant model and controllers

Silver-built controller models also provide access to all internal signals and all internal parameters of the Simulink models. This provides very valuable support for analysis, debugging and calibration of controllers.

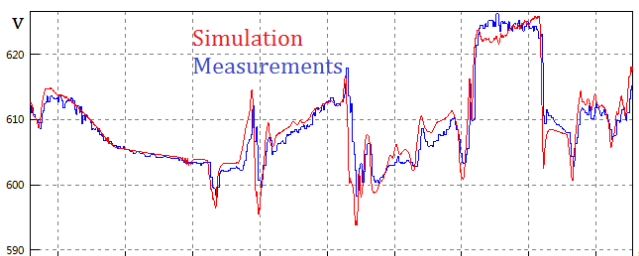
## 4 Validation/calibration of models

### 4.1 BMS – MCU system validation

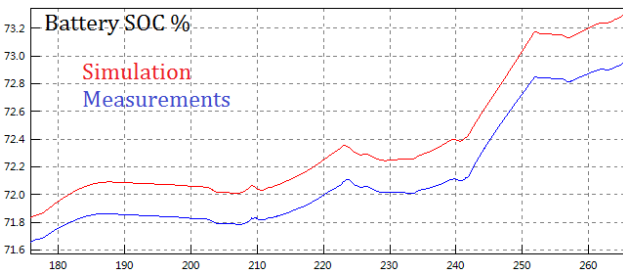
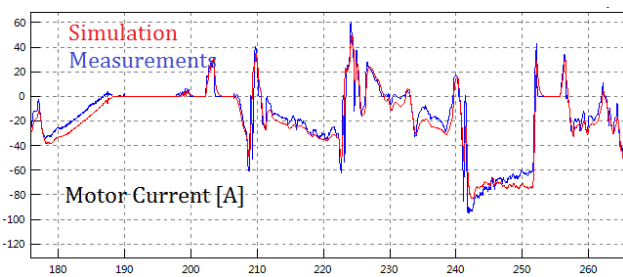
The High voltage circuit of BMS, MCU and DC accessories load is validated independently by using comparison between in-vehicle measurements and simulation results. The model unknown/uncertain parameters are first calibrated. These unknown parameters are typically the polarization resistance and polarization capacity and some MCU control software properties. The model used for this calibration/validation work is shown in Figure 14. Boundary conditions such as motor speed and motor torque target are imposed according to vehicle measurements. Some results are presented in Figure 15 and Figure 16.



**Figure 14: High voltage Electrical system validation model**



**Figure 15 : Battery voltage, comparison of in-vehicle measurement and simulation results**



**Figure 16 : Comparison of in-vehicle measurements and MCU/BMS currents and Battery State Of Charge**  
 The results for the BMS/MCU validation gives a very good agreement for the MCU current, battery SOC and battery voltage, including transients effects on the battery voltage. Difference between simulated SOC and recorded SOC is a static offset due to initial value of SOC.

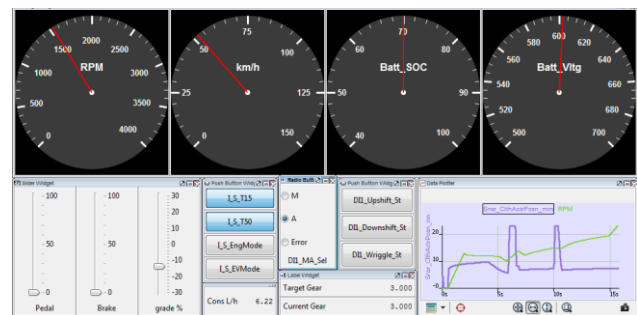
#### 4.2 Complete vehicle simulation

A special attention was given first to increase simulation speed. For instance unnecessary events in the exported FMU were removed, some high frequencies dynamics were neglected because irrelevant for the

work at hand. However, since we are using variable step solvers, quantifying exactly simulation speed is not possible since it will depend on the driving case. We give however rough estimates:

- When the vehicle is idle or in steady-state without gearshifts, simulation runs 20x faster than real-time.
- When the driving sequence involves numerous gearshifts, simulation runs 3x faster than real-time.

We give in Figure 17 a partial view of the graphical controls and displays used in the QTronic Silver, where the plant model, HCU and TCU are integrated together, with around 250 signals exchanged at 10ms time cycles between controllers and plant. The complete vehicle validation is still under progress at the day of writing.



**Figure 17 : Extract of dashboard/instruments for interactive simulation in QTronic Silver**

## 5 Applications and benefits

The above virtual powertrain will be used for several applications that we list below.

### 5.1 Control software development support

The virtual powertrain in QTronic Silver has very desirable properties to support the work of function developers for the TCU and HCU.

#### Accurate plant models from SimulationX

The plant models from SimulationX are accurate and represent the overall behavior of the complete vehicle, including all CAN signals and details of EMS, BMS, MCU control logic. The controllers can then be accurately checked and tuned against such plant models.



### Integration of calibration and measurements

Silver provides easy integration of in-vehicle measurements standard formats. This allows test engineers to provide data to function developers. Function developers can easily analyze problems met in the vehicle, define fix and corrections, and verify effects in the simulation.

### Integration of actual production c-code

During this project, we are currently integrating floating-point Simulink models. A next step will be the integration of actual production c-code, with fixed point arithmetic, tasks, along with A2L definition. The final production code used in the TCU and HCU can thus be fully tested, using Virtual ECUs.

### Control software Debugging in Silver

Simulink models built from Silver can access all internal signals and parameters of a model. This provides powerful function for debugging issues in control logic, arguably more comfortable and efficient than natively in Simulink.

## 5.2 Large coverage testing

Once an accurate/realistic virtual powertrain is available, it is possible to use it for large coverage testing using TestWeaver. QTronic TestWeaver can generate 1000's of test scenarios overnight and find problems and issues in the system. This increases system quality and system safety early in the development cycle. Compared to hand-written test scripts, TestWeaver scenario generation is a systematic process that will explore a large number of combinations and states.

In the case of the hybrid truck under consideration, state coverage objectives include gearshifts and hybrid mode transitions.

## 5.3 System optimization

The virtual powertrain established in the project can be used for system optimization, including control software optimization and calibration optimization. This work is efficiently supported since any function developer can introduce modifications and quickly get results on the new implementation. Moreover, Silver also supports scripting so that selected parameters can be automatically optimized using numerical optimization routines.

## 6 Conclusions

A virtual powertrain for a hybrid truck is being established in collaboration with Dongfeng Commercial Vehicles. For creating this system, several tools are being used, including ITI SimulationX, QTronic Silver and Matlab/Simulink.

The Virtual powertrain provides accurate simulation of the complete vehicle, down to gearshift synchronization events. Battery SOC and e-motor currents have been validated against in-vehicle measurements.

This Virtual Powertrain is applied for system optimization, HCU/TCU control logic debugging and large coverage testing with TestWeaver. These tasks can be conducted efficiently by engineers using PC-based deployable simulation.

## References

- [1] [www.itisim.com](http://www.itisim.com).
- [2] A.Abel, T.Blochwitz, A.Eichberger et al. Functional Mock-up Interface in Mechatronic gearshift simulation for commercial vehicles, 9<sup>th</sup> International Modelica Conference, 2012, Munich.
- [3] A.Junghanns, R.Serway, T.Liebezeit, M.Bonin. Building virtual ECUs quickly and economically. ATZ Elektronik, 03/2012, Volume7.
- [4] E.Chrisofakis, A.Junghanns. Simulation-based development of automotive control software with Modelica. Dresden: Modelica international conference, 2011.
- [5] M.Tatar, Schaich, Breitingner. Automated test of the AMG speedshift DCT control software. Berlin: 9th CTI Innovative Automotive Transmissions Symposium, 2010.
- [6] J.Mauss, M.Simons. Chip simulation of automotive ECUs. 9<sup>th</sup> symposium Steuerungssysteme fur automobile Antriebe, 2012, Berlin
- [7] N.Papakonstantinou, S.Klinger, M.Tatar. Test-driven development of DCT Control Software. 8th International CTI Symposium Innovative Automotive Transmissions, 2009, Berlin.
- [8] M.Neumann, M.Nass, M.Tatar. Absicherung von Steuerungssoftware fur Hybridsysteme, Autoreg 2011, Friedrichshafen.
- [9] L.Belmon, J.Yan. Modeling and simulation of DCT gearshifting for real-time and high-fidelity

analysis. SAE China-FISITA conference,  
F2012-C04-014, 2012

- [10] A.Abel, U.Schreiber, Valsania, Fornelli. Simulation based design of gearboxes for high-performance sports cars. Modena:11th HTCES conference, 2005.

# General fault triggering architecture to trigger model faults in Modelica using a standardized blockset

F.L.J. van der Linden, German Aerospace Center (DLR),  
Institute of System Dynamics and Control.  
Münchener Straße 20, 82234 Weßling, Germany  
Franciscus.vanderlinden@dlr.de

## Abstract

The implementation of faults in Modelica is currently not standardized, which leads to many non-compatible implementations. To support the standardization of fault implementations, a new standard for fault implementation and triggering is proposed. The proposed standard can handle parameter faults as well as variable faults during a time simulation to cover all common fault possibilities. Using instance modifiers as well as an inner-outer broadcasting method, the faults can be triggered in a central block. Furthermore, care was taken so that the simulation of the models in a fault-free condition can be guaranteed. A library using the proposed standard was developed. In this library, the fault implementation as well as the triggering of these faults was modeled with the end user in mind. An example implementation is presented which shows the capabilities of the library.

*Keywords: Failure, Fault, Modeling, Standardization, Fault Injection*

## 1 Introduction

Failure detection and health monitoring systems to improve reliability and lower maintenance costs become increasingly important. Therefore the design and testing of these algorithms need good prediction models combined with an efficient way to trigger all fault cases.

Implementing faults in Modelica models is no new terrain. Many different implementations of real systems have been made. For example Schallert [8] did a reliability and safety assessment. The faults are automatically identified based on parameter names. To set the parameters of the failed parts, a function is used which automatically sets

the parameters before simulation. Gao et al. [3, 2] did a fault analysis of electrical systems. To trigger the faults, two different methods are used; hard coding a fault in the model as well as creating a completely new model for a fault. Cui et al. [1] modeled an actuator system with automatically triggered faults. This automatic triggering is based on the predefined fault probability, but cannot be directly controlled. These works are all examples where faults are triggered in a Modelica implementation. However, all of these implementations use different ways to trigger the faults. Since there is a lack of standards implementation ways, all users must find a way for them self to trigger the faults.

Another approach for model-based diagnosis is used by RODON [7]. Uncertainty intervals for the model parameters combined with behavioral models are used to trigger faults. However, time simulations are not supported which limits its use in many applications. Also FaultWeaver [6] can be used to trigger faults in Modelica. It uses a set of models in Modelica. An external (non-Modelica) program is used to set the faults in Modelica and simulate the results.

In this paper, a set of **standardized** fault-output blocks is proposed in Section 3. These blocks use a designated data type to clearly identify these blocks as special fault blocks for further processing. Using these blocks, it is possible to create component models which include optional faults by the user. Care has been taken to make sure that all possible faults can be modeled by a single or a combination of standardized fault blocks. By analyzing the complete model, built from individual sub-models, a wrapper package can be automatically generated which can be used to activate all faults (Section 5). Care has been taken that it is possible to completely eliminate the fault

code from a model to increase simulation performance if not all faults are triggered. Furthermore, also quick model testing without a fault setup is possible.

The proposed implementation is based on a Dymola implementation making use of the Abstract Syntax Tree (AST) functions from the `ModelManagement` library. By using the proposed **open** and **standardized** fault blocks with a specialized fault type, it is possible to create similar functionalities in all Modelica solvers.

## 2 Fault injection demands for Modelica

To create a general environment to trigger faults in Modelica, care must be taken that all possible faults can be modeled using the proposed blocks. To make sure all possible faults are covered, a trade off study has been carried out. Fault implementations in the Modelica language can be generalized into classes. The following sections will highlight the different fault classes.

### 2.1 Fault variability classes

For Modelica usage, two different classes of faults can be identified:

1. Faults that have a very low time constant with respect to the simulation horizon and can be considered constant.
2. Faults with time constants faster than the simulation horizon which will cause transient behavior.

To further clarify the classes, a more detailed description including examples of each fault class is given.

#### 2.1.1 Parameter faults

The parameter fault class consists of faults that have a low time constant compared with the simulation horizon. Usually these faults are characterized by slow changes in time such as the variation of the viscosity of oil due to an aging process in a transmission application. Another example of a fault with a slow time constant compared to the simulation time are some high frequency electronics simulations. In these simulations, the temperature of the environment can often be characterized

as constant. Some examples of parameter faults are:

- Gear play
- Degradation of capacitors or batteries
- Oil viscosity degradation in a transmission
- Environment temperature increase in a fast switching application

Due to the very slow nature of these faults with respect to their simulation time, it is not necessary to have the possibility to model fault transients.

#### 2.1.2 Variable faults

The second class of faults are variable faults. These faults are characterized by the possibility that they can significantly change during a typical simulation. Quite often the study of a transient response is one of the main purposes of the simulation of such faults. Some examples are:

- Semiconductor short circuit
- Breakage of hydraulic oil line
- Gearbox tooth breakage
- Screw jam

The faults in this class can vary during a simulation run, and can cause a dynamic system response which might be of interest for the engineer.

### 2.2 Fault data type classes

The faults of both classes described in Section 2.1, can be divided into three types to represent the different cases needed to model faults.

#### 2.2.1 "On-Off" faults (Boolean)

On off faults are marked by having only two discrete states. Examples are jamming of a nutscrew and disconnection of electrical cables.

#### 2.2.2 Case faults (Integer)

Case faults are marked by having multiple discrete failure modes. An good example is a semiconductor failure:

- Normal operation
- Short circuit
- Open loop

	Constant	Variable	Flag value	Description
<b>On-Off Mode</b>	Increased friction	Screw jam	0	fault deactivated
<b>Continuous</b>	Bearing fault mode	Transistor	1 (default)	standard fault mode activated
	Gear play	Oil loss	2,3,...	optional extra fault modes

Table 1: Combined fault possibilities for Modelica models with examples. The choice between a variable and parameter fault is not always directly clear, and may need to be chosen as constant or variable based on the length of the simulation

### 2.2.3 Continuous fault (Real)

A continuous fault is a fault without an explicit discrete value. Examples are:

- Oil degradation
- Increased friction in a bearing
- Capacitor degradation

Combining the fault classes (Section 2.1) and the fault class properties (Section 2.2), six different combinations of faults are identified (see Table 1). These possibilities can model all general and advanced faults. In the next sections, the implementation as well as some extra features for easier fault handling and simulation performance are discussed.

## 2.3 Variable mode selection

To accommodate the reconfiguration of a model with a variable fault, a mechanism to decide if the fault can be activated during simulation must be implemented. This reconfiguration can be necessary to increase simulation speed in case of no failure or to switch between different failure modes. In the case of a parameter fault, this is known by definition. However in the case of a variable fault, this is not known. To be able to reconfigure such a model, it is therefore necessary to add a parameter signal flag which can be used to reconfigure the model. For maximal flexibility, it is chosen to add a mode selection using an integer constant as a flag. This flag can be used to reconfigure a model to include or exclude a fault. How to use the values of the flag can be seen in Table 2.

The same effect as the mode selection is possible by combining a parameter integer fault with an variable fault. However, combining two fault inputs for one fault makes it hard to use consistent naming.

Table 2: Variable mode selection flag

## 3 Fault triggering standardisation architecture

Defining faults types is not sufficient to define a usable Modelica implementation. For a good user-friendly implementation a well designed architecture is vital. Different ways to set up a fault triggering method are analyzed and their benefits compared.

### 3.1 Fault Architecture

Controlling of the faults in a global model using components with faults can be done in many different ways. Different ways are studied in this section and it is decided which methods are selected for the proposed standard.

To assess the overall performance of these methods, a set of criteria is defined to evaluate several important aspects for fault triggering. The implementation effort of setting up the general architecture is not evaluated as this effort has to be invested only once in the generation of the fault library.

These criteria are:

- Non physical connections:** The connections between the models should be based on physical quantities. Faults do not have a physical connections as they are triggered by wear, or external influences which are usually not modeled.
- Ease of implementation:** Effort for user to create a model from instances using faults (e.g. the development of a multistage gearbox using predefined faulty gearbox instances)
- Maintainability:** Effort to maintain a set of models with faults. Typical tasks would be adding or removing fault cases, restructuring models and keeping a well documented set of models
- Standardization:** Standardization effort to keep models compatible between different

business partners.

- (e) **Transients:** Possibility to model transients used for variable faults (see for a description Section 2.1.2).

Four methods to implement faults in Modelica have been tried out and analyzed. The results are assessed using the criteria (a through e).

1. **Model parameters:** Each fault is controlled by a parameter in the model. It is possible to "pull" these parameters up to the top level model. Also direct changing of the instance parameter in the model is possible. If the parameter is flagged appropriately, it is possible to create a system with automated parameter detection and central setting of the parameters. Such a structure does not need non-physical connections, is easy to implement for the user and, if a proper automation is used, can be well maintained. Also standardization using the proposed flag methods is possible. Since in this method it is only possible to handle parameters, no transients can be used.
2. **Model inputs:** Inputs are used to control faults in the models. By connecting the input connectors, it is possible to create a central element to control all faults. This method is often used for small fault systems. However, it leads to non physical connections between the models. Due to the high customization, maintainability and standardization cannot be guaranteed. The ease of implementation is good in small systems maintained by a single person, but quickly becomes more and more problematic as the model grows. Transients can be handled well.
3. **Bus system:** A bus system to connect faults. All fault signals are connected to a bus system. This way is similar to the model inputs, except that all faults are organized in one block. A bus system leads to non physical connections. The ease of implementation and maintainability depends highly on the complexity of the model, small systems can be easily implemented and managed, but it become quickly confusing. The standardization is better than using a direct model input since all faults are now marked in one fault-bus. However, still no automated algorithms can be used as it is impossible to properly

define a standard input. Transients can be handled well as it is possible to connect variables to a bus.

4. **Broadcasting:** Using an inner-outer structure, the fault models can obtain their values from a centralized point in the global model. Using an automated routine, all appropriate flagged faults can be found and managed in one central point. No non-physical connections are needed. If standardized, flagged and predefined fault blocks are used, the ease of implementation and maintainability is high. Also the standardization can be guaranteed by flagging the models. Using an inner-outer structure it is also possible to use transients. However, when only parameter Faults are used, this way of modeling is over complex and will always need a full setup of the variable faults. In contrast, it is possible to leave most parameter faults at their standard value and set only one fault without setting up a complete fault system.

In Table 3, the previously discussed four different approaches (1:4) are assessed using the criteria (a:e). From this analysis follows that the usage of model parameters (1) and a broadcasting system (4) have most advantages. It is therefore chosen to use following architecture:

- **Model parameters** to handle **constant** faults
- **Broadcasting system** for **variable** faults

## 4 Standardized fault class definition

All faults in a model must be recognized by automated scripts, while at the same time the user should have the freedom to name the model faults arbitrarily. To do so, special fault classes have been designed. This has the advantage that a fault is identified by the class name, and can be integrated without special care of instance names by the user. These fault classes are released under the Modelica 2 License.

Below the all fault classes are defined. For a parameter fault of the type Real, the type is defined in Code 1.

Description	(a) Non physical connections	(b) Ease of implementation	(c) Maintainability	(d) Standardization	(e) Transients
(1) Model parameters	+	+	+	+	-
(2) Model inputs	-	±	-	-	+
(3) Bus system	-	±	±	±	+
(4) Broadcasting system	+	±	+	+	+

Table 3: Fault triggering approaches. A detailed description of the criteria can be found in Section 3.1

Code 1: Real parameter fault

```
type Parameter_Fault_Real =
  Real "Value of the Real Fault";
```

Using this special fault class for each Real fault, it is possible to clearly identify each instance of this model as a Real parameter fault.

The same is done for variable faults. Since these faults are more complex, a record with three parameters is used. In Code 2 the definition of this Fault is shown.

Code 2: Real variable fault

```
record Variable_Fault_Real
  "External Fault Triggering parameters"
  Boolean externalFaultOn=false
  "External fault controlling
  (true = global)";
  Integer faultIndex = 1
  "External fault index";
  Integer faultMode = 1
  "Optional fault mode for model
  reconfiguration";
end Variable_Fault_Real;
```

The first Boolean `externalFaultOn` is used to switch between the local default fault definitions and external global control. The integer `faultIndex` is used to set the channel in the external global fault triggering (see Section 5). The Integer `faultMode` is used to set the optional fault mode selection as discussed in Section 2.3.

The examples given in this section are for Real faults. The code for Integer and Boolean faults can be found in Appendix A.

Using these class definitions, it is possible to set up a complete fault triggering system. In Section 5 an implementation for Dymola using the Model-Management toolbox is presented. Since these defined faultclasses are open and standardized, algorithms or plug-ins for programs can be developed by users, also for other Modelica solvers.

## 5 FaultTriggering library

Beside the definition of a standard, a library has been built to support the user with implementing faults. Using the definitions from Section 4, blocks are created which simplify the implementation of faults in a model. Two versions of these outputs are made; one for textual modeling and one for usage in the diagram layer. Also a method to manage the fault signals in a single block is proposed.

In Figure 1, an overview of the final fault setting structure is given. In the generated wrapper model, it is possible to set the parameter and variable faults. The parameters are set in an automatically generated structure (see Section 5.3.2) and the variable faults are handled using a generated bus system (see Section 5.3.3).

### 5.1 Parameter fault modeling

The textual modeling block for a parameter fault is a simple block with a parameter of type `Parameter_Fault_Real` (for Real faults). By extending this block in the model, a parameter fault is directly correctly implemented and its name is `constRealFault`.

In Code 3 the code for the Real parameter fault is given. Parts of the complete path to the components are abbreviated for a better overview. Integer and Boolean faults are implemented using the same approach.

Code 3: Real parameter fault for textual modeling

```
block InternalConstantRealFault
  "Generate constant Fault of type Real"
  extends ...Icons.RealFault;
  parameter ...Types.Parameter_Fault_Real
    constRealFault= 1
    "Constant output value";
end InternalConstantRealFault;
```

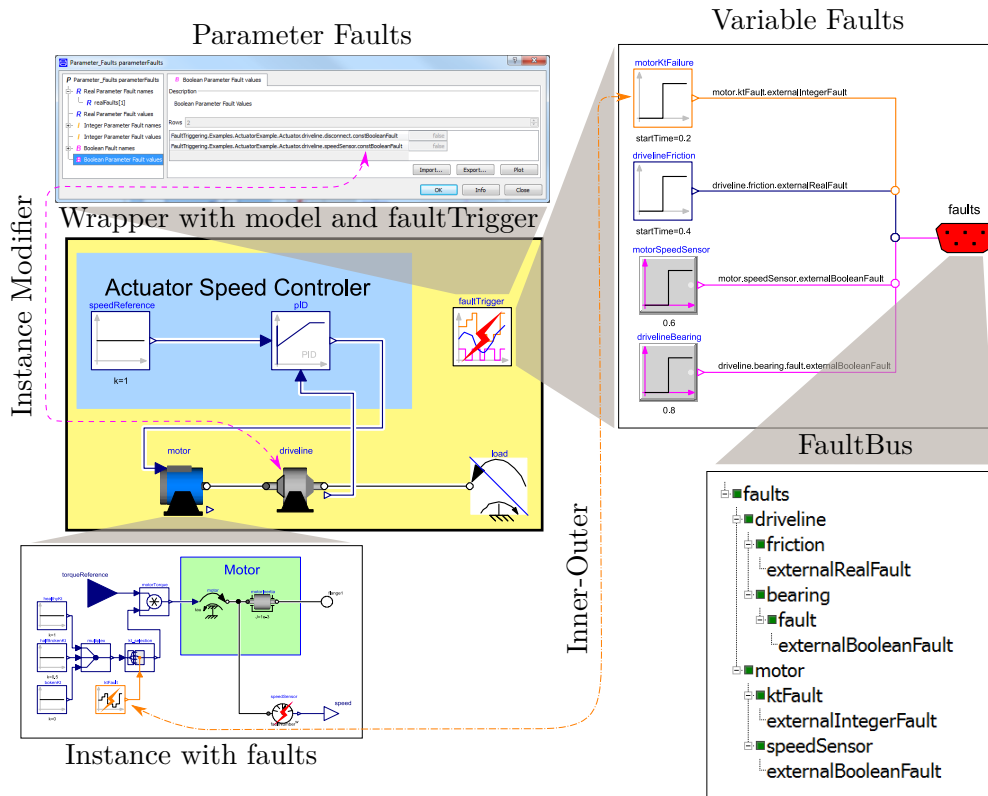


Figure 1: Automatically generated wrapper model (yellow) which contains the extended original model and the block `faultTrigger`. In this block all parameter and variable faults can be set. The parameter faults communicate directly with the model instances using instance modifiers (pink dash-dotted line), the variable faults using a bus system connected to a global inner/ outer system (orange dashed line).

The blocks for graphical modeling environment are extensions of the discussed textual modeling approach and the interface `Modelica.Blocks.Interfaces.S0`. This creates a block (see Code 4) with a single Real output whose value is set by the fault parameter.

Code 4: Real parameter fault for graphical modeling

```

block ConstantRealFault
  "Generate constant signal of type Real"
  extends Modelica.Blocks.Interfaces.S0;
  extends ...InternalConstantRealFault;
equation
  y = constRealFault;
end ConstantRealFault;
    
```

## 5.2 Variable faults

The variable fault blocks are more complicated to implement than the parameter fault blocks. These blocks need the information from a central block in which the fault signal is defined. To do so an inner-outer structure has been set up to communicate the fault signals. In this section, first the global

block will be discussed followed by the local fault blocks.

### 5.2.1 Global variable faults control

For each variable fault (Real, Integer and Boolean), a single channel is reserved in a variable with  $n$  channels (with  $n$  the number of faults of each type). This variable is defined in a central `FaultTrigger` block extended from `...FaultOutput.Partial_FaultTrigger`. Each fault can be coupled to fault sources using modelica code in this global block. This can be done by hand or an automated script which is proposed in Section 5.3.3. The partial model can be seen in Code 5. This model is defined as "inner" in the annotations, so that the local fault injection blocks can communicate with this block.

Code 5: Partial model for variable fault input framework

```

partial model Partial_FaultTrigger
  "partial model defining fault classes"
  parameter Integer realFaultSize
    "Number of real fault channels";
    
```



```

parameter Integer integerFaultSize
  "Number of integer fault channels";
parameter Integer booleanFaultSize
  "Number of boolean fault channels";

Real    realFault[realFaultSize]
  "Real Fault trigger";
Integer integerFault[integerFaultSize]
  "Integer Fault trigger";
Boolean booleanFault[booleanFaultSize]
  "Boolean Fault trigger";

annotation (
  defaultComponentPrefixes="inner")
end Partial_FaultTrigger;

```

## 5.2.2 Variable fault modeling classes

The variable fault models get their signals from the global fault control model. In Code 6 the code for a variable fault is given. In this model, the variable `fault` is the actual fault value. Each fault uses its own fault channel in the variables `realFault`, `integerFault` and `booleanFault`. To select which channel is to be used from these variables, the parameter `faultNumber` is defined. This parameter is generally set by an automated system (see Section 5.3.3).

To be able to directly operate a model with variable faults in the model for testing purposes, a parameter with a default fault value is defined in `fault_local`. Using the switch (`externalRealFault.externalFaultOn`), the local fault can be changed to the value defined in the global fault block. Using this, it is guaranteed that each block has a valid output without setting up a global fault block.

Code 6: Real variable fault for textual modeling

```

block InternalRealFault
  "Generate variable Fault of type Real"
  extends ...Icons.RealFault;
  outer FaultTrigger faultTrigger;
  parameter Real fault_local = 1
    "Default fault value if no external
    triggering is used";
  parameter ...Types.Variable_Fault_Real
    externalRealFault =
    ...Types.Variable_Fault_Real()
    "External Fault Triggering parameters";
  Modelica.Blocks.Interfaces.RealOutput
    fault "Final fault value";

protected
  ...Types.Fault_SelectRealFault
    faultNumber;
equation

```

```

faultNumber =
  externalRealFault.faultIndex;
fault =
  if externalRealFault.externalFaultOn
  then
    faultTrigger.realFault[faultNumber]
  else fault_local;
end InternalRealFault;

```

The faults for graphical modeling are made by extending Code 6 in a model with two outputs (Code 7): One real output for the fault signal and one optional integer output for the mode signal.

Code 7: Real variable fault for graphical modeling

```

block VariableRealFault
  "Generate variable signal of type Real"
  extends ...Internal.InternalRealFault;
  parameter Boolean useModelModeSelection
    "toggles external mode selection";
  Modelica.Blocks.Interfaces.RealOutput y;
  Modelica.Blocks.Interfaces.IntegerOutput
    mode = externalRealFault.faultMode
    if useModelModeSelection
    "Connector of Integer output signal";
  equation
    y = fault;
  end VariableRealFault;

```

## 5.3 Automated fault handling

To keep overview of the faults in a model and help the user with fault channel selection for each fault, an automated fault handling algorithm is developed. This algorithm can detect the parameter and variable faults in the selected model. Also all faults in the instances used in this model can be detected. Setting and internal handling of these faults is different for parameter and variable faults. A library is automatically generated which contains a wrapper model that extends the original model. Also a central block to manage the faults is instantiated in this wrapper model. In this block, the configuration of the parameter and variable faults is handled.

### 5.3.1 Automated fault finding

Using the Dymola `ModelManagement` toolbox, it is possible to investigate a model with its sub-models. Using these features, it is possible to generate a model tree from a model with all instances. A schematical example of such a model tree can be found in Figure 2. Using the type definitions from Section 4, all faults in a model can be found and

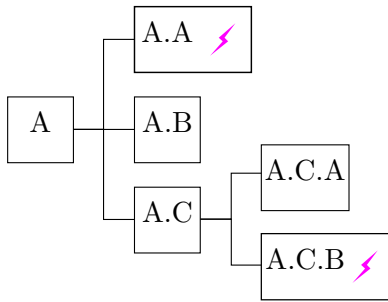


Figure 2: Model tree with faulty models on several levels. Blocks with a lightning symbol (A.A & A.C.B) are extensions of the standardized fault classes.

classified. Also the "path" to the model can be found. The complete path of the fault instance `bearing_stuck` in the model `Actuator` will be represented as `Actuator.bearing_stuck`. In the following sections, the generated fault tree and fault classification will be used in the implementation of the library features.

### 5.3.2 Global parameter setting

All the parameter faults can be found and identified using a method described in Section 5.3.1. It is possible to directly change these values by using an instance modifier generated by the fault-search algorithm by hand.

However, in case of large models with many faults or many different cases to analyze, this can quickly become unclear and tedious. To aid the user, a structure is automatically generated using the scripts supplied in the library which includes all faults together with their default values. This structure is used as a parameter in the global `faultTriggering` model. These values are automatically linked to the instance modifiers in the wrapper model. By creating different fault structures, fault cases can be defined. Each fault structure stands for a clearly defined simulation case.

### 5.3.3 Global variable setting

To aid the user with setting the variable faults, a hierarchical `faultbus` is generated from the fault structure (see Figure 1). It is possible to directly connect the fault source signals to the hierarchical bus. The hierarchical bus system itself is connected to the `realFault`, `integerFault` and `booleanFault` variables (see Section 5.2.2). The corresponding fault index is automatically set in

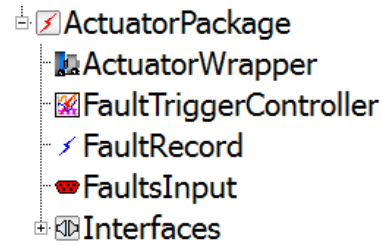


Figure 3: Automatically generated Fault library

the model using component modifiers. Using this approach, mistakes with mixing up the channels are not possible, as this is automated. Also the use of an automatically generated bus makes connecting the fault sources easy.

In Figure 3, a generated library is shown with its default components.

## 6 Examples

To test the library functionality a simple actuator model is built consisting of a motor with PID control, a simple driveline and a load. The total model has 6 faults: 2 parameter faults, and 4 variable faults. Using the fault processing algorithms presented in Section 5.3, a package is generated. The model wrapper adds the `faultTrigger` block in which all faults can be set. In this block all fault inputs are defined. The variable faults are set in the block of type `FaultTriggerController` (instance `faultTrigger`). Using the parameter record in this block, it is possible to set all parameter faults. An overview of this functionality is shown in Figure 1.

The result of a simulation with progressive faults is shown in Figure 4. The dynamic effects of a breaking component can be seen by the changing response of motor speed and torque. By changing or duplicating the `faultTrigger` block, it is possible to create multiple fault scenarios for a single model. The original model stays unchanged and can be used for all analysis, healthy as well as broken.

## 7 Conclusion and Discussion

In this paper, a method to standardize the implementation of faulty components in Modelica is specified. It is possible to implement parameter, as well as varying fault signals. The code for the proposed faulty components is included to aid the

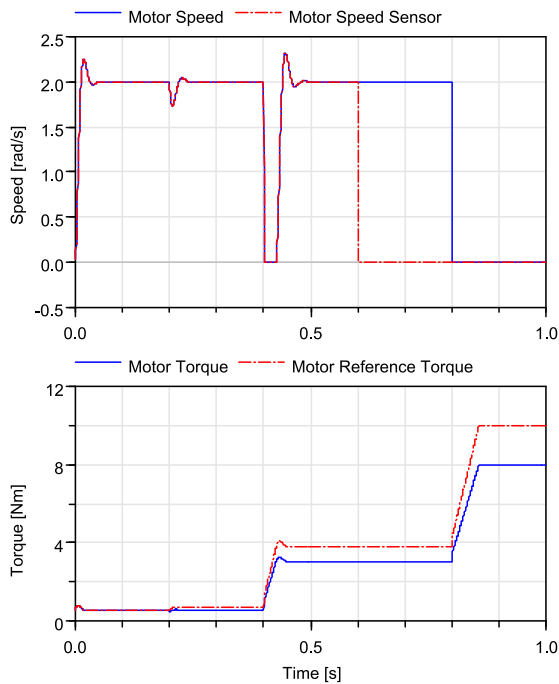


Figure 4: Results of a simulation with progressive faults. At  $t=0.2$ s the motor constant drops, at  $t=0.4$ s, an increased friction in the driveline is triggered, at  $t=0.6$ s, the speed sensor of the motor breaks and finally at  $t=0.8$ s the driveline bearing gets stuck.

standardisation of fault implementation in Modelica.

Moreover a library has been created which supports the user to set these faults by automatic generation of a wrapper library. This wrapper includes all parameter faults in a parameter structure and a bus system to connect the variable faults. This functionality is enabled by the implementation of a search algorithm to search the model for the standardized fault classes.

An example model has been built and the methods to implement the faults in the model have been proved valuable. At the moment the proposed Fault Library is used in the Actuator EMA library [4, 5, 9]. The standardization of these faults has led to an easy implementation process. The model designer can focus on implementing the faults in the model without paying attention to the interfaces and the compatibility between the models.

## Acknowledgement

The research leading to these results has received funding from the European Union's Seventh

Framework Program (FP7-284916) for ACTUATION 2015 under grant agreement no. 284915.

## A Modelica Code for Faults

The code for the implementation of the fault classes is given in this section. Using strictly this code, it is possible for automated fault systems to search for all faults in a model.

### A.1 Real faults

Code 8: Real parameter fault

```
type Parameter_Fault_Real =
  Real "Value of the Real Fault";
```

Code 9: Real variable fault

```
record Variable_Fault_Real
  "External Fault Triggering parameters"
  Boolean externalFaultOn=false
  "External fault controlling
  (true = global)";
  Integer faultIndex = 1
  "External fault index";
  Integer faultMode = 1
  "Optional fault mode for model
  reconfiguration";
end Variable_Fault_Real;
```

### A.2 Integer faults

Code 10: Integer parameter fault

```
type Parameter_Fault_Integer =
  Integer "Value of the Integer Fault";
```

Code 11: Integer variable fault

```
record Variable_Fault_Integer
  "External Fault Triggering parameters"
  Boolean externalFaultOn=false
  "External fault controlling
  (true = global)";
  Integer faultIndex = 1
  "External fault index";
  Integer faultMode = 1
  "Optional fault mode for model
  reconfiguration";
end Variable_Fault_Integer;
```

### A.3 Boolean faults

## Code 12: Boolean parameter fault

```
type Parameter_Fault_Boolean =
  Boolean "Value of the Boolean Fault";
```

## Code 13: Boolean variable fault

```
record Variable_Fault_Boolean
  "External Fault Triggering parameters"
  Boolean externalFaultOn=false
  "External fault controlling
    (true = global)";
  Integer faultIndex = 1
  "External fault index";
  Integer faultMode = 1
  "Optional fault mode for model
    reconfiguration";
end Variable_Fault_Boolean;
```

## References

- [1] CUI, X., MA, J., AND ZENG, S. The fault modeling methodology of actuator system based on Modelica. *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety* (June 2011), 997–1002.
- [2] GAO, J., JI, Y., BALS, J., AND KENNEL, R. Fault Detection of Power Electronic Circuit using Wavelet Analysis in Modelica. In *Proceedings of the 9th International MODELICA Conference* (Munich, Germany, Sept. 2012), no. 76, pp. 513–522.
- [3] GAO, M., HU, N., QIN, G., AND XIA, L. Modeling and fault simulation of propellant filling system based on Modelica/Dymola. *2008 2nd International Symposium on Systems and Control in Aerospace and Astronautics* (Dec. 2008), 1–5.
- [4] GIANGRANDE, P., HILL, C., GERADA, C., AND BOZHKO, S. Multi-Level Library of Electrical Machines for Aerospace Applications. In *Proceedings of the 10th International Modelica Conference* (2014).
- [5] HILL, C., GIANGRANDE, P., GERADA, C., AND BOZHKO, S. Implementation of a Multi-Level Power Electronic Inverter Library in Modelica. In *Proceedings of the 10th International Modelica Conference* (2014).
- [6] JUNGHANNS, A., MAUSS, J., AND TATAR, M. TestWeaver - A Tool for Simulation-based Test of Mechatronic Designs. In *Proceedings of the 6th International Modelica Conference* (2008), pp. 341–348.
- [7] LUNDE, K. Object-oriented modeling in model-based diagnosis. *Proceedings of Modelica Workshop, Lund, Sweden* (2000), 111–118.
- [8] SCHALLERT, C. Inclusion of Reliability and Safety Analysis Methods in Modelica. In *Inclusion of Reliability and Safety Analysis Methods in Modelica* (June 2011), pp. 616–627.
- [9] VAN DER LINDEN, F., SCHLEGEL, C., CHRISTMANN, M., REGULA, G., HILL, C., GIANGRANDE, P., MARÉ, J.-C., AND EGAÑA, I. Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters. In *Proceedings of the 10th International Modelica Conference* (2014).

# Using Fault Augmented Modelica Models for Diagnostics

Raj Minhas   Johan de Kleer   Ion Matei   Bhaskar Saha  
Bill Janssen Daniel G. Bobrow   Tolga Kurtoglu  
Palo Alto Research Center  
3333 Coyote Hill Road, Palo Alto, CA 94304 USA  
contact email: dekleer@parc.com

## Abstract

We propose a model-based diagnosis framework in which Modelica models of faulted behavior are used in combination with a Bayesian approach. The fault augmented models are automatically generated through a process developed as part of our Fault Augmented Model Extension (FAME) work. Fault diagnosis using a Bayesian approach is based on computing a set of probability density functions, a process that is usually intractable for any reasonably complex system. We use Approximate Bayesian Computation (ABC) to bound the numerical and computational complexity. The basic idea is to use fault augmented Modelica models to create probability distributions of possible outcomes and then compare those distributions against actual observations to perform parameter estimation. The detection of faults is treated as a model selection problem and the inference of their severity levels is treated as parameter estimation. The diagnostic precision of this approach is evaluated on a Modelica vehicle drive line model.

*Keywords: fault models; diagnosis; machine learning; model translation; bayesian methods*

## 1 Introduction

Modelica [6] is an object-oriented, declarative, multi-domain modeling language for component-oriented modeling of complex systems. It is used to execute numerical simulations to determine the behavior of a system. This approach frees the designer to efficiently explore a wide set of designs to see which meets customer requirements, without needing to physically construct the systems.

In addition to predicting behaviors through numerical simulations of Modelica models, we propose to use the same simulation models for diagnosis. Modelica's focus on simulation would seem to make it a

poor choice for diagnosis. After all, diagnosis is the inverse of simulation. Simulation predicts the behavior of a system given a (correct) model. Diagnosis must infer how the model has changed (i.e., faulted) from observed behavior.

Most model-based diagnosis algorithms [4, 10] perform inference on declarative models. Although Modelica supports the writing of declarative models, too many Modelica models (including many in the MSL) contain imperative constructs making direct application of existing model-based diagnosis algorithms problematic. RODON [2] is a Modelica inspired approach to modeling, but Modelica models first have to be re-written by hand in qualitative declarative form. We know of no system identification or FDI technique that applies for DAE models with boolean constraints (as Modelica models translate into). Our approach, on the other hand, applies on Modelica models directly no matter what types of constraints they contain.

This paper presents a framework for model-based diagnosis in which Modelica tools play a fundamental inference role in a Bayesian approach. Numerical simulations of Modelica models are used to build statistical models for the behavior of a system for all of its fault-operating modes; statistical models that help determine a diagnosis solution based on observations.

The simulation of a system under different fault-operating modes is enabled by fault-augmented Modelica libraries. As part of our Fault Augmented Model Extension (FAME) work, we have developed an approach [5] for automatic model construction of Modelica fault models. Our model fault-augmentation approach is based on analyzing Modelica libraries for fault susceptibility and on modifying them to enable simulation of faulty components. FAME provides the set of possible faulted behaviors for each component (e.g., a resistor might be open, shorted, or resistance shifted by some undetermined amount). FAME can use as input statistical models from reliability analysis that determine the fault activations and the amount

of change in the values of the parameters.

Consider a grossly simplified vehicle model illustrated in the Modelica model of Figure 1. The brake may be working normally (Nominal state) or may be having friction related faults (Slipping or Sticking). The severity of these fault modes is modeled in Modelica by a parameter called  $\text{amount} \in [0, 1]$  where a value of 0 indicates no fault (i.e., Nominal state) and a value of 1 indicates total failure. The part will likely be unusable (and hence considered faulty) for values far less than 1.

When the brake absolute angular velocity  $\omega$  is not zero, the frictional torque applied by the brake is a function of a velocity dependent friction coefficient  $\mu(\omega)$ , the normal force  $f_n$ , and of a geometry constant  $c_{geo}$ , which takes into account the geometry of the device and the assumptions on the friction distributions:

$$\tau = c_{geo} \times \mu(\omega) \times f_n.$$

Our approach can be used to estimate such parameters in addition to performing diagnostics. For this exercise, we assume that the only unknown parameter is the `amount` but we could just as easily estimate the other parameters such as  $c_{geo}$  and  $f_n$  from the observed data.

## 2 FAME approach

We have developed faultable models for the models in the Modelica Standard Library. A detailed description of FAME can be found in [5], while [8] describes how it can be used in a design-tool chain to perform reliability analysis. Here we summarize the essential details of FAME for the purpose of diagnosis. Every model class definition which contains faults is replaced with a new class definition, a Modelica model class subsuming the original model class and adding declarative behavior to allow simulation of the faults.

An encapsulated enumerated type is defined, listing the various fault modes of the class, along with the Nominal mode. A discrete mode parameter of this new type is introduced, defining the mode in which an instance of the class is operating. An if-equation similar to Figure 2 is added, so that each operating mode can define its own dynamics.

The set of equations which apply in each fault mode is expressed in the appropriate branch of this if-equation. The process also flattens the superclasses of the model into the rewritten class, and introduces two new externally visible parameters, `mode` and

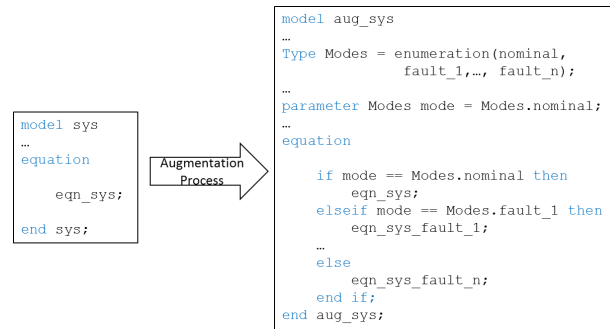


Figure 2: Alternative dynamics are enabled for each operating mode.

`amount`, as well as an enumerated type giving the possible faults for this component, `Modes`.

Modelica models are augmented with two types of faults: *power faults* and *parametric faults*. Power faults model loss of power at the connection points and is implemented through the addition of new components abstractly called “dampers” that implement this behavior. We show below how the `Stick` fault is implemented at `flange_a` of the `Brake` component:

```
model Brake
...
Modelica.Mechanics.Rotational.
    Interfaces.Flange_a flange_a;
FAME.Dampers.RotationalWithConnectEquations
    _famefault_flange_a(amount=0.0);
...
end Brake

model RotationalWithConnectEquations
input Real amount(min=0.0, max=1.0);
encapsulated type Modes = enumeration(
    Nominal,
    Stick,
    Broken);
parameter Modes mode = Modes.Nominal;
Modelica.Mechanics.Rotational.
    Interfaces.Flange_a port_a;
Modelica.Mechanics.Rotational.
    Interfaces.Flange_a port_b;
equation
...
elseif mode == Modes.Stick then
    port_b.tau + port_a.tau =
        (1/max(Constants.verySmall, 1-amount)-1)
        *der(port_a.phi);
    port_b.phi = port_a.phi;
else
    ...
end if;
end RotationalWithConnectEquations;
```

We note that a new component called `_famefault_flange_a` was added to the `Brake` model; component that implements the behavior corresponding to the `Stick` fault.

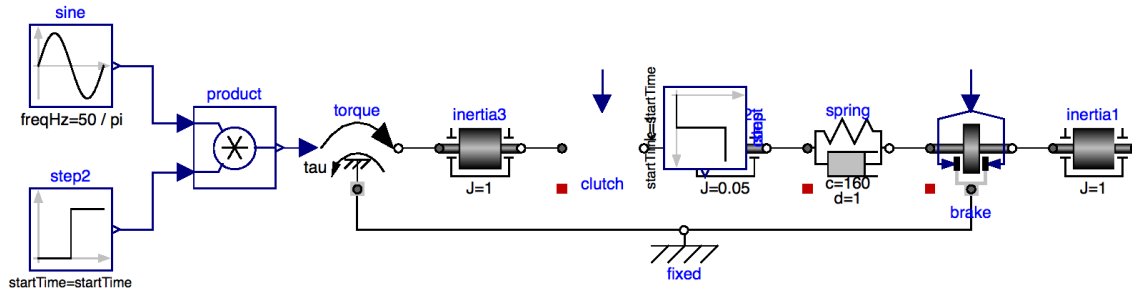


Figure 1: Simple drive line.

Similarly, parametric faults are handled by introducing a new component that defines the pattern of change for the parameter to which this component is associated. For example in the case of the Brake model, `_famefault_mue_pos` is an instance of the newly added `_famefaults_mue_pos` component that corresponds to the `mue_pos` parameter:

```

model Brake
  model _famefaults_mue_pos
    extends FAME.Parametric.
    BaseParametricFault(amount=0.0);
    type Modes = enumeration(
      Nominal,
      Slip);
    parameter Modes mode=Modes.Nominal;
    equation
    if mode==Modes.Slip then
      y = u*{{1,0},{0,1-amount}};
    else
      y = u;
    end if;
  end _famefaults_mue_pos;
  ...
  _famefaults_mue_pos _famefault_mue_pos
    (u=mue_pos,reddeclare type
      ParamType = Real[size(mue_pos,1),2]);
  ...
end Brake;
    
```

References to the original parameter are replaced with an expression which references to this new variable. For example, the parameter `mue_pos` is replaced in the fault-augmented version of the Brake by `_famefault_mue_pos.y`.

We consider several other fault modes. Consider the simple spring-damper system of Figure 3. Figure 4

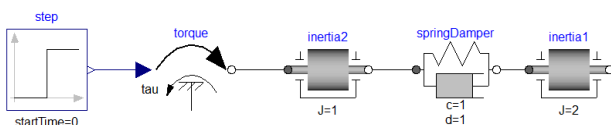


Figure 3: Spring Damper example.

shows three simulation results for the Nominal,

Stick and Broken modes for `inertial1`. The underlying intuition of our approach is to pre-compute many simulations under many fault scenarios and perform on-line diagnosis by identifying the pre-computed simulation results which best matches the observation. In the remainder of this paper we describe how this intuition has been implemented.

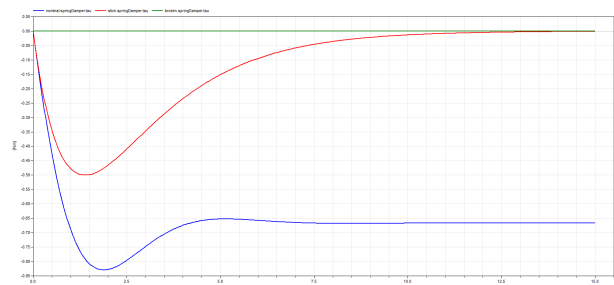


Figure 4: Nominal, Stick and Broken operating modes for the spring-damper system

### 3 Fault detection in the drivetrain system

We demonstrate our proposed approach by detecting faults in the drivetrain system shown in Figure 1. We model five failure modes: brake slipping, brake sticking, clutch slipping, clutch sticking, and spring sticking. The failure mode of a stuck brake is modeled in the FAME library as a dynamic damper component that adds damping equal to the `amount`<sup>1</sup>. This has the effect of increasing the relative friction in the flanges which results in a loss of power. The failure mode of a slipping brake is modeled as a decrease in the velocity dependent friction coefficient by an amount equal to the severity of the fault:

$$\mu(\omega) \mapsto \mu(\omega) \times (1 - \text{amount}).$$

<sup>1</sup>Under normal operating conditions, the `amount=0` so the system does not experience this additional damping.

This has the effect of reducing the friction torque<sup>2</sup>. The other failure modes are modeled in a similar manner. Our aim is to infer from observed data whether the system is exhibiting a fault mode — if so, we also need to infer the severity of the fault (i.e., the value of amount). We achieve this by using ideas developed for Approximate Bayesian Computation (ABC).

## 4 Approximate Bayesian Computation

Modelica models are formal representations of (hybrid) differential equations (DAEs). Existing results on fault diagnosis of DAEs are not able to cope with such mathematical models in their generality. They either focus on the structural aspect of the system, ignoring its dynamics [7], or they make simplifying assumptions, such as linearity, on the DAE model [3]. Therefore more practical approaches for fault diagnosis must be employed.

In this section, we give an overview of the ABC method, based on material from [9, 12, 1] — see those references for more details. The typical tasks are to estimate unknown model parameters and to do model selection. Let  $M$  be a model parameterized by some parameters  $\Theta$  whose joint prior density is  $\pi(\Theta)$ . Given data  $d$  generated by the model  $M$ , we are interested in estimating the posterior probability  $\pi(\Theta|d)$ . From Bayes rule, we know that

$$\pi(\Theta|d) \propto f(d|\Theta)\pi(\Theta),$$

where  $f(d|\Theta)$  is the likelihood of the data given the parameters. The prior probability of the parameters is known so we need to compute the likelihood of the data in order to estimate the posterior probability of the parameters. A similar approach can be used for model selection. Let  $M_1$  and  $M_2$  be two models and we would like to infer which of them is more likely to have generated the given data  $d$ . Using the Bayesian analysis framework, we compute the Bayes factor  $B_{12}$  to summarize the evidence provided by the data for model  $M_1$  over model  $M_2$ :

$$B_{12} = \frac{P(M_1|d)/P(M_2|d)}{P(M_1)/P(M_2)},$$

where  $P(M_i)$  is the prior probability of model  $M_i$  and  $P(M_i|d)$  is the posterior probability of the model  $M_i$  given the data  $d$ . A value of  $B_{12}$  between 1 and 3

suggests weak evidence in favor of  $M_1$ , a value between 3 and 20 suggests positive evidence, a value between 20 and 150 suggests strong evidence, and a value greater than 150 suggests very strong evidence. The prior probabilities of the models don't depend on the given data and can be pre-computed. So computing the Bayes factor comes down to computing the ratio of the posterior probabilities of the models given the data:

$$B_{12} \propto \frac{P(M_1|d)}{P(M_2|d)} \propto \frac{f(d|\Theta_1)}{f(d|\Theta_2)}.$$

For any reasonably complex model, the likelihood computation is intractable so ABC approaches like rejection are used to approximate it. In order to simplify the problem computationally, it is common to define a function  $f_s: \mathbb{R}^n \rightarrow \mathbb{R}^m$  that maps the given data  $d \in \mathbb{R}^n$  to some representative statistic  $s \in \mathbb{R}^m$  where  $m \ll n$ . Then we define a distance metric  $dist_s: \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$  to measure closeness of two sets of statistics. Let  $\hat{d}_{M_i}$  be simulated data generated from a model  $M_i$ .

### 4.1 Rejection Technique

If  $dist_s(f_s(d), f_s(\hat{d}_{M_i})) < \varepsilon$  for some threshold  $\varepsilon$  then the simulated data is accepted for the given observed data  $d$  — otherwise, it is rejected. Now assume that we generate  $N$  data sets simulated from the model  $M_i$  as follows. For each of the  $N$  iterations, draw a parameter vector  $\Theta_i$  from the prior distribution  $\pi(\Theta_i)$  and simulate data  $d_s^i$  from  $M_i$ . Assume that  $\bar{N}$  of those simulated data sets are accepted given a threshold  $\varepsilon$ . Then we can approximate  $P(M_i|d)$  as  $\frac{\bar{N}}{N}$ . The distribution of the  $\Theta_i$  for each of the accepted iteration approaches  $\pi(\Theta_i|d)$ . The approximated values approach the true values asymptotically as  $N \rightarrow \infty$  and  $\varepsilon \rightarrow 0$  if the statistics are sufficient to describe the data. The downside of this approach is that if  $\varepsilon$  is small then  $N$  needs to be high in order to achieve a reasonable approximation. In other words, it can be very computationally expensive.

### 4.2 Classification Technique (alternative approach)

For model selection, rather than computing the ratio of posterior probabilities, we could use a classification approach instead. Here we treat the model indicator  $i$  as the response variable and the summary statistics as the dependent variables. Any standard classification technique such as multinomial logistic regression, random forest, neural network etc. can be used

<sup>2</sup>Again, under normal operating condition, the amount=0 so the system does not experience this loss in torque.



to train a model based on the simulation data.<sup>3</sup> The trained model is evaluated on the statistics  $f_s(d)$  of the observed data to directly estimate  $P(M_i|d)$ . This approach may be needed for more complex diagnostic tasks, but as the rejection approach performs so well, we use it in our example.

## 5 Evaluating the approach

The first step in ABC is to generate a large number of simulations. To constrain the problem appropriately, we assume that the observed output of the system is generated in response to a known input. For example, the response of a dynamical system to a step input (called a step response) is typically used to reason about the system behavior and may be appropriate for diagnosis as well. This choice is part of the feature selection and needs to be made at the time of diagnostic design. For each of the five fault modes, we generate 10,000 step-response simulations<sup>4</sup> — we first sample the `amount` from its prior distribution<sup>5</sup> and then use that value to perform the simulation. This was done by parameterizing the variables in Modelica file — a snippet of the Modelica model is shown below — the string `FAULT_AMOUNT` is replaced with the sampled value before running a simulation.

```
Brake brake(fnmax = 1600,
  _famefault_mue_pos(mode = Modes.Slip,
    amount=FAULT_AMOUNT) ;
```

The second step in ABC is to compare observed data against the simulated data. For computational reasons, we compare features computed from the simulated and actual data rather than comparing the raw data directly. For this exercise, we evaluate the step response of the drive train system and compute the following features of the absolute angular velocity of the inertial load connected to the brake: (1) mean, (2) maximum, (3) 25<sup>th</sup> percentile, (4) 50<sup>th</sup> percentile, (5) 75<sup>th</sup> percentile, (6) inter-quartile range, and (7) time to go to zero. The values of these seven features can be thought of as a vector of dimension seven. The difference between the observed vector and the simulated vectors is used to compute an estimate of the

<sup>3</sup>Any technique that returns a normalized measure of classification should be usable.

<sup>4</sup>For a more complex model, we may need to generate a higher number of simulations.

<sup>5</sup>For this example, we sample `amount` from a uniform distribution:  $\text{amount} \sim U(0.003, 0.5)$ . In practice, the choice of this distribution will depend on the belief of the designers about the distribution of the fault amount - such a distribution may be learnt from field performance data of similar systems. This is a typical design choice in Bayesian analysis and a non-informative distribution such as a uniform distribution may be used if no other source of information is available.

likelihood that the observed value was generated from the simulated distribution. For more details, please see [11].

In order to evaluate the effectiveness of our approach, we measure the diagnostic accuracy of detection the following 11 faults.

1. *Brake Slipping* fault mode with `amount=0.1`
2. *Brake Slipping* fault mode with `amount=0.25`
3. *Brake Sticking* fault mode with `amount=0.1`
4. *Brake Sticking* fault mode with `amount=0.25`
5. *Clutch Slipping* fault mode with `amount=0.1`
6. *Clutch Slipping* fault mode with `amount=0.25`
7. *Clutch Sticking* fault mode with `amount=0.1`
8. *Clutch Sticking* fault mode with `amount=0.25`
9. *Spring Sticking* fault mode with `amount=0.1`
10. *Spring Sticking* fault mode with `amount=0.25`
11. *Nominal* mode (i.e., with no fault mode or a fault mode with a very small `amount`)

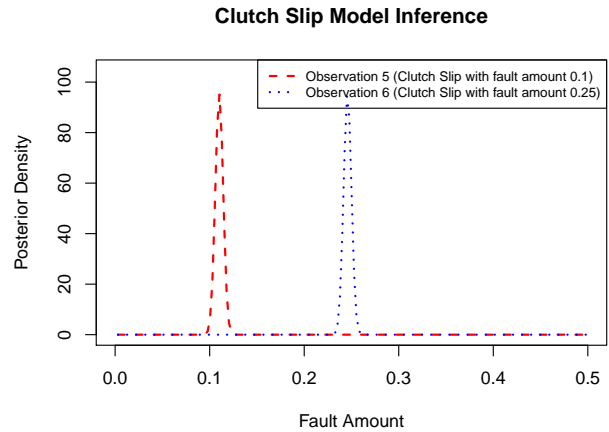
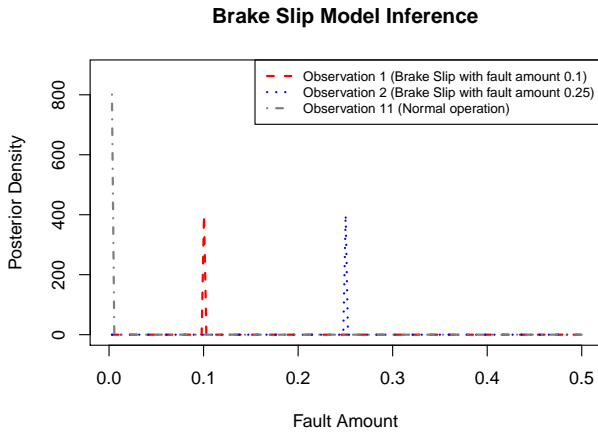
For each of these faults, we infer the fault model that was most likely to have generated it and estimate the `amount`. The analysis was done using the *ABCtoolbox* suite [11].

## Results

We first show how each model fares against the observed data and then put it all together to generate the final diagnosis.

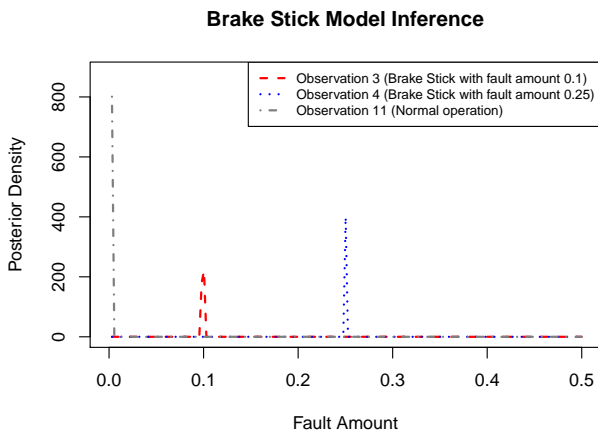
### *Brake Slipping* model

When the eleven faulty behaviors are compared against the simulations from the *Brake Slipping* model, the marginal distribution of the model is nearly zero for all but faults #1, #2 and #11. The graphs below show the posterior density distributions of `amount` for those three faults — as the graph shows, the inference is correct and the estimate of the `amount` is also very accurate. Of course, in order to make the final diagnosis for a fault, the marginal density for this model will need to be compared against the densities for the other models.



**Brake Sticking model**

When the eleven faulty observations are compared against the simulations from the *Brake Sticking* model, the marginal distribution of the model is nearly zero for all but faults #3, #4, and #11. The graph below shows the posterior probability distributions of amount for those three faults — again, the inference is correct and the estimate of the amount is also very accurate.

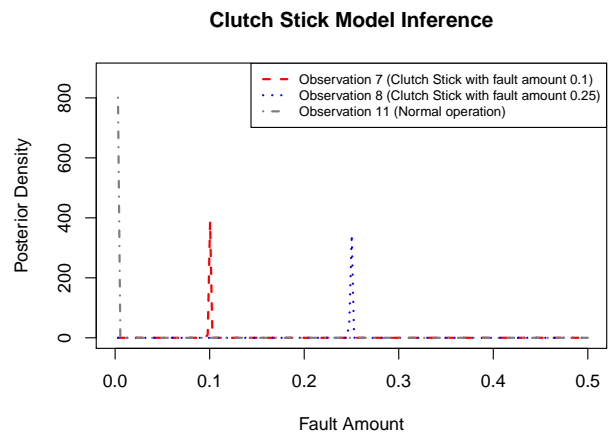


**Clutch Slipping model**

When the eleven faulty observations are compared against the simulations from the *Clutch Slipping* model, the marginal distribution of the model is nearly zero for all but faults #5, and #6. The graph below shows the posterior probability distributions of amount for those two faults — while the distributions are not as peaked as the ones for brakes, the inference is still correct and the mode of the distribution is over the correct value of amount.

**Clutch Sticking model**

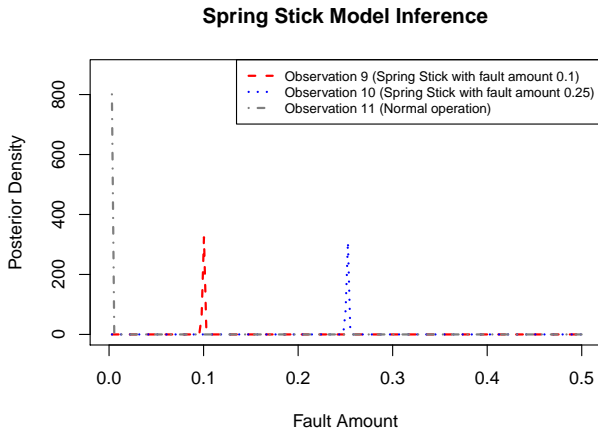
When the eleven faulty observations are compared against the simulations from the *Clutch Sticking* model, the marginal distribution of the model is nearly zero for all but faults #3, #7, #8, #9, and #11. In this case, the marginal distribution is non-zero for two faults (#3 and #9) that do not correspond to the clutch sticking failure mode. However, the overall diagnosis still turns out to be correct because the marginal distribution numbers for the correct models are much higher than these numbers (see Section 5). The graph below shows the posterior probability distributions of amount for faulty observations #7, #8, and #11 — as in the other cases, the posterior distributions of amount have very narrow peaks over the correct values.



**Spring Sticking model**

When the eleven faulty observations are compared against the simulations from the *Spring Sticking* model, the marginal distribution of the model is nearly

zero for all but faults #9, #10, and #11. The graphs below show the posterior probability distributions of amount for those faults — again, both the fault mode inferences and the estimates of amount are correct.



### Final diagnosis

As mentioned previously, all the faulty observations except #3, #9 and #11 are inferred to have a single (and the correct) cause. While fault #11 (which corresponds to the normal operation) has significant ambiguity regarding the cause, the inferred amount is always nearly zero in all those cases — this correctly indicates the absence of any fault mode. In other words, fault #11 is correctly associated with normal operation. Fault #3 is inferred to have been generated by either *Brake Sticking* or *Clutch Sticking* failure modes. Similarly, fault #9 is inferred to have been generated by either *Clutch Sticking* or *Spring Sticking* failure modes. So for these two cases, we need to look at the ratios of the respective posterior densities, i.e., compute the Bayes factor in order to complete the diagnosis (see Section 4 for more details).

**Fault #3** For this fault, the marginal posterior density of *Brake Sticking* model is  $9.8 \times 10^9$  while that of *Clutch Sticking* model is 1.7. So the Bayes factor for *Brake Sticking* is  $\frac{9.8 \times 10^9}{1.7} = 5.7 \times 10^9$  which is very strong evidence in favor of *Brake Sticking* (i.e., the correct diagnosis).

**Fault #9** For this fault, the marginal posterior density of *Clutch Sticking* model is  $3.6 \times 10^3$  while that of *Spring Sticking* model is  $2.9 \times 10^8$ . So the Bayes factor for *Spring Sticking* is  $\frac{2.9 \times 10^8}{3.6 \times 10^3} = 8 \times 10^4$  which is very strong evidence in favor of *Spring Sticking* (i.e., the correct diagnosis).

So the FAME based inference approach is able to make the correct diagnosis for all the faults.

## 6 Final remarks

This paper has demonstrated that a straight-forward application of machine learning techniques to simulation can be used to accurately diagnose systems. In fact, it can diagnose systems even if some of the (non-faulted) parameters are unknown. This approach has some inherent limitations: (1) the expansion to multiple faults will require exponentially more pre-computed simulations and therefore would only scale to simple systems, (2) active diagnosis will require deriving features for many system variables which may be impractical, (3) the features (i.e. sufficient statistics of the signal) we use are somewhat determined by the requirements of the system and must be determined at the outset, and (4) if the system can have a wide variety of exogenous inputs, too many pre-computed simulations will be required in order to diagnose for each possible input stream.

The expansion to multiple faults can be ameliorated somewhat by the fact that the simulations are done offline and can be easily parallelized. This complexity may be further managed if a reasonable assumption can be made about an upper limit on the number of simultaneous faults (thereby reducing the complexity from exponential to polynomial).

## 7 Acknowledgments

This work was partially sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program. Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

## References

- [1] M. G. B. Blum and O. Francois. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(20):63–73, 2010.
- [2] Peter Bunus and Karin Lunde. Supporting model-based diagnostics with equation-based object oriented languages. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*, Paphos, Cyprus, July 2008.

```
model Spring
  "Linear 1D rotational spring"

  // locally defined classes in Spring
  model _famefaults_flange_a = FAME.Dampers.RotationalWithoutConnectEquationsAasP;
  model _famefaults_flange_b = FAME.Dampers.RotationalWithoutConnectEquationsAasP;
  model _famefaults_c
    extends FAME.Parametric.BaseParametricFaultAmountAsParameter(amount=0.0);

  // locally defined classes in _famefaults_c

  type Modes = enumeration(Nominal, Fatigue);

  // components of _famefaults_c
  parameter Modes mode=Modes.Nominal;

  // algorithms and equations of _famefaults_c
  equation
    if mode==Modes.Fatigue then
      y = u*(1-amount);
    else
      y = u;
    end if;
  end _famefaults_c;

  // components of Spring
  Modelica.Mechanics.Rotational.Interfaces.Flange_a flange_a
    "Left flange of compliant 1-dim. rotational component";
  FAME.Dampers.RotationalWithoutConnectEquationsAasP _famefault_flange_a(amount=0.0);
  Modelica.Mechanics.Rotational.Interfaces.Flange_b flange_b
    "Right flange of compliant 1-dim. rotational component";
  FAME.Dampers.RotationalWithoutConnectEquationsAasP _famefault_flange_b(amount=0.0);
  parameter Modelica.SIunits.RotationalSpringConstant c(final min=0, start=1.0e5);
  _famefaults_c _famefault_c(u=c);
  Modelica.SIunits.Angle phi_rel(start=0)
    "Relative rotation angle (= flange_b.phi - flange_a.phi)";
  Modelica.SIunits.Torque tau "Torque between flanges (= flange_b.tau)";
  parameter Modelica.SIunits.Angle phi_rel0=0 "Unstretched spring angle";

  // algorithms and equations of Spring
  equation
    tau = _famefault_c.y*(phi_rel-phi_rel0);
    phi_rel = _famefault_flange_b.port_b.phi-_famefault_flange_a.port_b.phi;
    _famefault_flange_b.port_b.tau = tau;
    _famefault_flange_a.port_b.tau = -tau;
    connect(flange_a,_famefault_flange_a.port_a);
    connect(flange_b,_famefault_flange_b.port_a);
  end Spring;
```

Figure 5: Fault augmented model for rotational spring.

- [3] Wen Chen, M. Saif, and B. Shafai. Fault diagnosis in a class of differential-algebraic systems. In *American Control Conference, 2004. Proceedings of the 2004*, volume 5, pages 4398–4402 vol.5, 2004.
- [4] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.
- [5] Johan de Kleer, Bill Janssen, Daniel G. Bobrow, Tolga Kurtoglu, Bhaskar Saha, Nicholas R. Moore, and Saravan Sutharshana. Fault augmented modelica models. In *24th International Workshop on Principles of Diagnosis*, pages 71–78, Jerusalem, Israel, 2013.
- [6] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, Piscataway, NJ, 2004.
- [7] Mattias Krysander and Mattias Nyberg. xstructural analysis for fault diagnosis of dae systems utilizing graph theory and mss sets. Technical Report LiTH-ISY-R-2410, Department of Electrical Engineering, Linköping University, 2002.
- [8] Zsolt Lattmann, Adrian Pop, Johan de Kleer, Peter Fritzson, Bill Janssen, Sandeep Neema, Ted Bapty, Xenofon Koutsoukos, Matthew Klenk, Daniel Bobrow, Bhaskar Saha, and Tolga Kurtoglu. Verification and design exploration through meta tool integration with openmodelica. In *Proceedings of the 10th International Modelica Conference*, Lunde, Sweden, 2014.
- [9] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*, 16:1791–1798, 1991.
- [10] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.
- [11] D. Wegmann, C. Leuenberger, S. Neuenschwander, and L. Excoffier. Abctoolbox: a versatile for approximate bayesian algorithms. *BMC Bioinformatics*, 2(11):116, 1990.
- [12] W. Zhang and D. J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 27:2025–2035, 2002.



# From Modelica Models to Fault Diagnosis in Air Handling Units

Raymond Sterling<sup>1</sup>, Peter Struss<sup>2</sup>, Jesús Febres<sup>1</sup>, Umbreen Sabir<sup>2</sup>, Marcus M. Keane<sup>1</sup>

<sup>1</sup>Informatics Research Unit for Sustainable Engineering, Ryan Institute, NUI Galway, Ireland

<sup>2</sup>Computer Science Department, Technische Universität München

[raymond.sterling@nuigalway.ie](mailto:raymond.sterling@nuigalway.ie), [struss@in.tum.de](mailto:struss@in.tum.de)

## Abstract

This paper presents a methodology for model-based fault detection and diagnosis underpinned by modelica models and using a qualitative approach to diagnosis, which has been applied to diagnosis of an air handling unit based on data recorded by a building management system. The main steps from model development to component diagnosis are discussed and illustrated using a heating coil component.

*Keywords: model-based diagnosis, heating coil, calibration, fault detection and diagnosis*

## 1 Introduction

Heating Ventilation and Air conditioning (HVAC) systems are known for being very inefficient for different reasons, one of the most common causes being the presence of undetected failures in one or more of its components. Undetected faults can remain for long periods due to different factors: compensations made by the control algorithms of other elements belonging to the same system; lack of proper maintenance, improper timing of flow of energy to/from the building, etc. Even when systems are known to suboptimal operation, the presence of faults may be very difficult to manually localize and identify, making it a costly task for human operators who only act when indoor environmental conditions are not met. This lack of timely intervention raises the need for developing automated fault detection and diagnosis methods and technologies that assist the building operator.

Different fault detection and diagnosis (FDD) methodologies have been developed for HVAC systems, mostly based on expert knowledge to help identifying the faulty condition and its source [1]. However, a new trend in FDD is that of using models of the HVAC systems providing a base line for optimal operation, and supporting the detection of deviation from this optimum [2]. Model-based methods, offer the advantage of an increased flexibility to adapt to

different and innovative HVAC systems.

The focus of this paper is on a model-based diagnostic solution that uses a **qualitative model** for the part of the HVAC system corresponding to the **Air Handling Unit (AHU)**. This solution is derived from a general first-principle Modelica model and exploits a general diagnosis algorithm that isolates and identifies faults that occur frequently and can cause significant loss of system performance in AHUs: passing heating- and cooling-coil valves, and stuck dampers. An application example using a heating coil model is presented and provisions are made for the extension to other components.

The paper is structured as follows: section 2 provides an intuitive introduction to model-based diagnosis (MBD); section 3 outlines model requirements for on-line diagnosis while section 4 presents the modelica models and its calibration. In section 5 an example of the complete tool chain is discussed and finally, sections 6 and 7 provide concluding remarks and future work.

## 2 Model-Based Diagnosis: an intuitive introduction

Models used for designing and verifying control usually capture the nominal behaviour of the controlled physical system but are less reliable when modelling behaviours related to faulty operation. In fact, model-based diagnosis is able to perform fault **localization** using only models that represent the intended behaviour of the system (OK models) [3]. However, fault **identification** requires modelling the possible relevant faulty behaviours, as well, which may also lead to a more focused localization [3]. Therefore, for each system component, a health variable is defined as the health status (failure mode) of that component. A system health assignment (or health mode) is the set of health assignments for all components in the system.

To each component OK or failure mode, the respective (mis)behaviour is captured by a model (e.g. a set of (differential) equations. For example, a passing

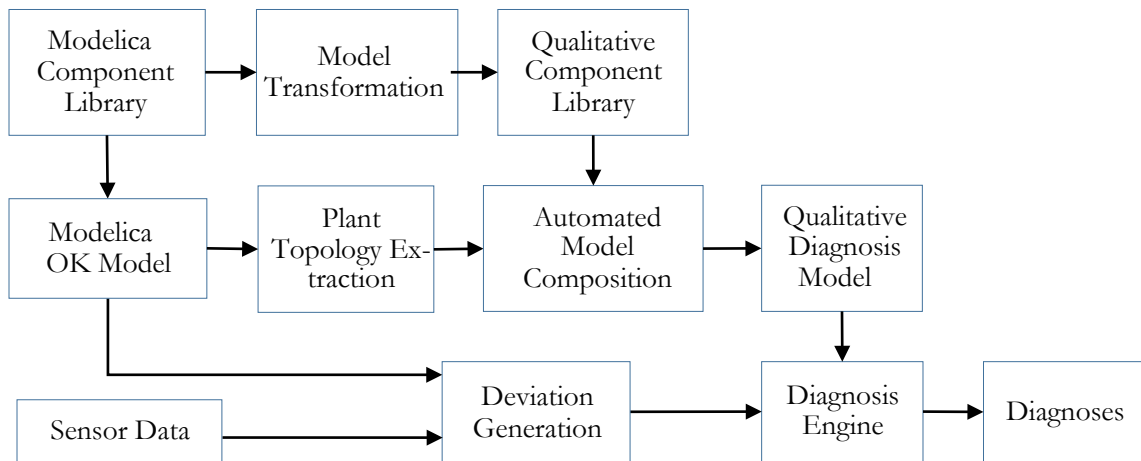


Figure 1 From model to diagnosis, the MBD chain

heating coil valve will have hot water flowing in the heating coil when the heating coil is supposed to be switched off, in which case the air heats up after passing over this heating coil. Hence, in this case, the control setting is heating coil valve closed, but with the fault, the valve is actually open.

Based on this, each system health assignment implies one behaviour model of the entire plant, which is obtained by aggregating the component (fault) models.

Model-based diagnosis is based on an explicit representation of the knowledge about the components and the information about the plant structure, which determines how the components interact with each other. Based on a library of generic component models and the representation of the plant topology, a system model (possibly covering both the nominal and faulty behaviours) can be obtained automatically. This model is exploited by a generic diagnosis algorithm, which is not plant-specific and even not domain-specific (Figure 2). This way, diagnostics tailored to a specific plant require only the specification of the plant structure and component models; they are generated automatically instead of being hand-tailored. For the purposes of this research work, a plant model consisting of component models was built manually and then fed to the diagnostics tool that produces the diagnostics system automatically. However, steps have been taken to automate the full process by parsing the modelica file. This parsing can be done as long as certain naming convention is in place.

### 3 From Model to On-line Diagnosis

In this section, we present a complete workflow and

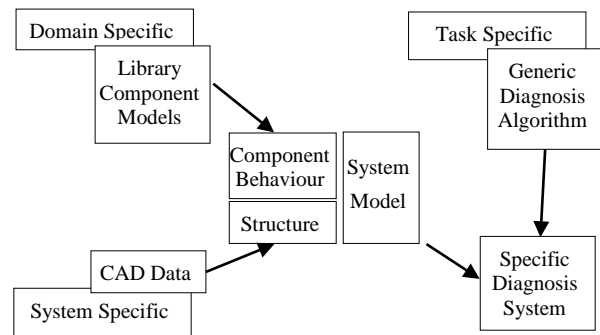


Figure 2 Generating Diagnostic Systems system modules required to build a diagnostic solution for a class of plants (AHU) and to deploy it for a single plant and run it on-line, which is illustrated in Figure 1. Here, we give only an overview of the steps and modules, the most important ones being discussed in more detail in the following sections.

- **Producing the general solution** involves:
  - the production of a library of Modelica models (section 4) and;
  - its transformation into a qualitative diagnostic model library (top row Figure 1).
- **Producing an application system** based on the general solution, requires
  - The configuration and calibration of a Modelica model of the correct behaviour (named OK model and explained in section 4);
  - the composition of the diagnostic model based on the diagnostic library and the component structure of the plant, which can be extracted from this Modelica system model. This composition step is part of the functionality of the tool used in this work, *Raz'r* (from



OCCM Software GmbH). The extraction of the component structure from Modelica has not yet been realized, but is expected to be straightforward, given that the models have been developed following certain requirements, which are stated in section 3.1.

- For **on-line diagnosis**,
  - qualitative deviations are generated by computing the difference between the real data (currently for steady state only) and the predictions generated by the OK model of the plant (implemented in Modelica), and determining qualitative deviations based on given thresholds. A steady state filter is used to extract steady state data from the real operation data. The resulting qualitative deviations of dependent variables (and zero deviations for the exogenous variables) are processed by
  - the runtime diagnosis engine, which is produced by the code generator from the *Raz'r* module by compiling the consistency-based diagnosis algorithm and the diagnostic plant model into very compact C-code. The output is the set of all mode assignments containing minimal combinations of component faults that are consistent with the abstract observations [4].

### 3.1 Requirements on Modelling for Model-Based Diagnosis

In order to support the model-based diagnosis approach as previously outlined, the diagnosis models and, hence, also the numerical models to generate them from have to satisfy particular requirements:

- Strictly **component-oriented** modelling: the library has to be organized around the component types (with models that can be parameterized) that constitute the plant and that are units subject to diagnosis, e.g. heat exchangers, mass exchanger, mass movers, etc.
- **Fault models** should be represented (perhaps with a parameter characterizing the fault, such as the opening of a passing valve)
- The plant model has to be configured strictly according to the **real physical interconnections** in the plant. It must not include computational artefacts that link certain variables that are not really interacting directly via a physical connection. This includes using the concept of connectors in Modelica to reflect the channels of physical interactions between components (rather than connections via single variables as, for instance, in Matlab/Simulink).

The models in the library have to be formulated in a **context-independent manner** and must not rely on implicit assumptions about the presence and correct functioning of other components, even though they may exist in most standard configurations. This is relevant for two reasons: it enables the re-use of the component models for different plants, and it is a precondition for the adequacy of the models in fault situations.

## 4 Modelling a Simple Heating Coil

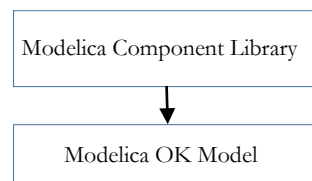


Figure 3. MBD first steps, model development and calibration

### 4.1 Model Development

Model development was driven by the specific application needs as specified in the previous section. These needs also encompass matching of the type of information interchanged between elements, reusability of the models, best use of manufacturer's data for setting up models and ease of use.

Ease of use and best use of manufacturer's data are closely related since the manufacturer's data is the first source of information a model developer will have in hand. In this regard, the developed model is such that this data is input into the parameters of the models corresponding this way to a first calibration step based on the manufacturer provided operation point. Table 1 shows the parameters, from the manufacturer's data, to be provided to the heating coil model.

Table 1. Manufacturer's datasheet operation point values needed as parameters for model setup

Heating Coil	air input temperature
	air output temperature
	air mass flow rate
	water input temperature
	water output temperature
	water mass flow rate

The heating coil model calculates the outlet steady-state conditions in both, water and air sides, using equations derived from the conservation of energy and mass principles and the definition of effectiveness in the classical eff-NTU method which given by equations (1), (2) and (3) [5]:

$$Q = C_a * (T_{ao} - T_{ai}) \tag{1}$$

$$Q = C_w * (T_{wi} - T_{wo}) \tag{2}$$

$$Q = eff * \min(C_a, C_w) * (T_{wi} - T_{ai}) \tag{3}$$

The effectiveness *eff* depends on the coil configuration (parallel flow, counter flow, or cross flow with both streams unmixed) [6]. The full modelica code is out of the scope of this paper but a snippet of the three main equations (1), (2), and (3) is shown below to illustrate the match between equation formulation and modelica code development:

```
Qflow = Cflow_a*(To_a - Ti_a);
Qflow = Cflow_w*(Ti_w - To_w);
Qflow=eff*min(Cflow_a,Cflow_w)*(Ti_w-
Ti_a);
```

For the heating-coil component, there are inputs and outputs for flow of air through the ducting, and flow of hot water through the heating coil. Hence, mass- and energy-balance equations must be defined for the airflow and water-flow. The imposition of energy- and mass- balance provides the remainder of the Modelica model equations.

### 4.2 Calibration

The calibration methodology uses real operation data obtained from the facility’s building management system (BMS). For the calibration procedure, instead of trying to adjust each of the component’s parameters, the approach used is by assuming all the calibration can be done with the valve model explained below in this section.

In the heating coil, the air outlet temperature is controlled by water mass flow rate using valves. A control signal determines the valve’s position.

Real valves have no linear behaviour but they may present non-linear behaviour and even hysteresis. To model the valve’s hysteresis, several options can be followed, e.g. using on-off hysteresis, linear hysteresis and non-linear hysteresis. For the purposes of this research work, a hysteresis as shown in Figure 4 was chosen since it produced a good trade-off between accuracy and simplicity. The chosen hysteresis model will still be a good representation of the real operation of the valve while it does not add important calculation burden to the model.

There are three parameters to calibrate. ‘*mflowMAX*’ is the water mass flow rate when the control signal is equal to 1 (maximum opening position), *centHys* and *delta* characterise the hysteresis’ curve and the on/off points.

The real data has to be carefully observed to find maximum opening points and then the *mflowMAX* value is fixed in order to decrease the difference between real data and model results of the controlled variable in those points (temperature and/or humidity ratio).

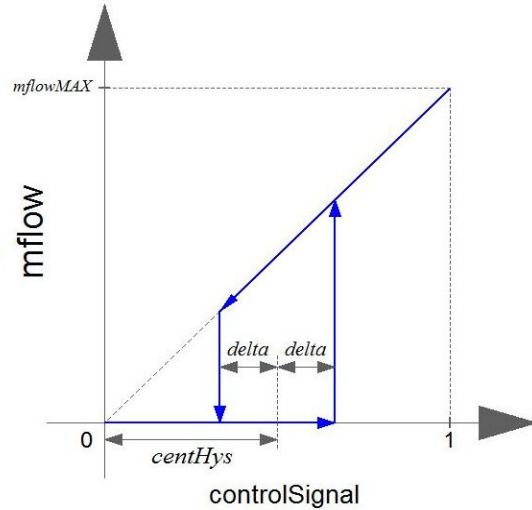


Figure 4 Valve hysteresis function

To determine *centHys* and *delta*, the employed strategy was to find sharp changes in controlled variable (output air temperature). When the controlled variable has a sharp raise, the control signal coincides with a value equal to *centHys+delta*; controlled variable has a sharp decrease, the control signal coincides with the value equal to *centHys-delta*.

Pre and post calibration results can be seen in Figure 5 and Figure 6.

In Table 2 we show calibration accuracy based on error metrics such as root mean square error (RMSE), coefficient of variation of the RMSE (CV-RMSE), mean bias error (MBE) and, normalised MBE (NMBE).

Table 2. Calibration Results

Heating Coil	RMSE (K)	CV RMSE	MBE (K)	NMBE
Pre-Calibration	1.57	0.52	-0.76	-0.26
Pos-Calibration	0.54	0.18	-0.07	-0.02

## 5 Qualitative Diagnostic Models

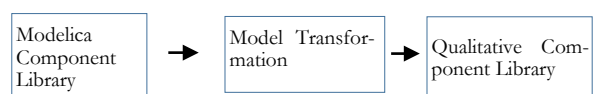


Figure 7 From numerical model to qualitative models

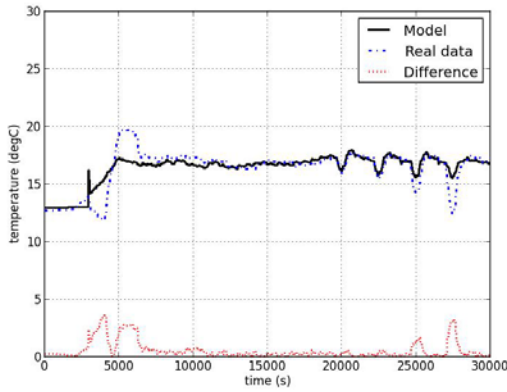


Figure 5 **non-calibrated** simulated (model) vs. measured (real) output air temperature for the heating coil model.

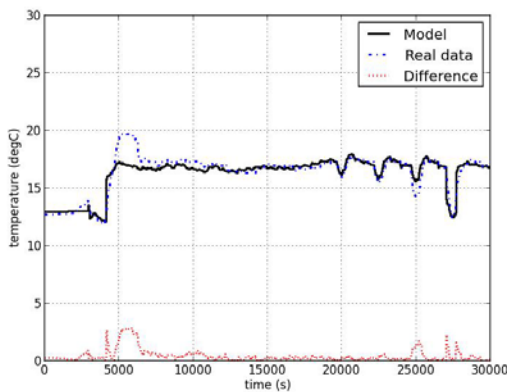


Figure 6 calibrated simulated (model) vs. measured (real) output air temperature for the **heating coil** model.

Creating a diagnostic library, based on the Modelica library, requires its transformation into a diagnostic model library. Figure 7 illustrates these steps.

The models used in our diagnostic approach are stated in relative, rather than absolute terms: they capture the deviation of variable values from the respective under nominal behaviour.

Following [7], [8]: the qualitative deviation of a variable  $x$  is defined as:

$$\Delta x := \text{sign}(x_{\text{act}} - x_{\text{nom}}) \quad (4)$$

Equation (4), captures whether an actual (observed, assumed, or inferred) value is greater, less or equal to the nominal value. The latter is the value to be expected under nominal behaviour, technically: the value implied by the model in which all components are in OK mode.

Qualitative deviation models can be obtained from standard models stated in terms of (differential) equations by canonical transformations, such as equations (5) and (6). We use  $\oplus$ ,  $\ominus$  and  $\otimes$ , to denote addition, subtraction and multiplication on signs.

$$a + b = c \Rightarrow \Delta a \oplus \Delta b = \Delta c \quad (5)$$

$$a * b = c \Rightarrow (a_{\text{act}} \otimes \Delta b) \oplus (b_{\text{act}} \otimes \Delta a) \ominus (\Delta a \otimes \Delta b) = \Delta c \quad (6)$$

It is important to note that these equations do not contain and require values for the reference values  $x_{\text{nom}}$  and, hence, can be applied to different plants and under distinct operating modes. The qualitative deviation models, obtained from the Modelica models, reflect current modelling assumptions, (steady state, and no deviation in airflow) and become very compact due to their qualitative nature and because constants can be dropped and just replaced by their signs. Internally, this model is automatically transformed into an efficient data structure representing finite relation.

In the following, we illustrate how this transformation can be done by manipulating the equations. According to energy balance equations (equations (1), (2) and, (3)), and assuming no losses, the energy balance in equation (7) can be reformulated in terms of deviations ( $\Delta$ ) as in equation (8).

Assuming that the air flow and the water temperature (drop) are positive and not deviating and replacing the capacity flow by the mass flow  $m\text{flow}_w$  (which differ only by a constant factor), we obtain equation (9) which applies to all modes of the coil.

$$0 = C_a * (T_{a0} - T_{a1}) - C_w * (T_{w1} - T_{w0}) \quad (7)$$

$$0 = \Delta (C_a * (T_{a1} - T_{a0})) \oplus \Delta (C_w * (T_{w1} - T_{w0})) \quad (8)$$

$$0 = \Delta T_{a1} \ominus \Delta T_{a0} \oplus \Delta m\text{flow}_w \quad (9)$$

Following equation (4), each of the variables used for diagnostics (equation (9)) can have a deviation of the measured value from the simulated one as follows:

- positively ('+'), when the actual (measured, predicted, or assumed) value is above the simulated plus a threshold;
- negatively ('-'), when the actual value is below the simulated minus a threshold;
- or not deviate ('0'), when the actual value is within the simulated value plus/minus the threshold.

Table 3 depicts the resulting relation on the three deviation variables, i.e. all solution tuples of equation (9). For instance, the first three rows of the table indicate the intuitive fact that, if the mass flow shows no deviation, a deviation of the incoming air temperature

will simply be propagated to the output air temperature.

On the other hand, a positive deviation of the output air temperature in combination with no deviation in the input air temperature, is only consistent with a positive deviation in the mass flow rate of the water (last-but-one row). From the diagnostic perspective, this reveals a fault in the coil (e.g. a passing valve), because a correct coil will not produce a deviating water flow. A valve stuck closed may lead to a negative deviation “-“, if the command Cmd to the valve is “open” (to some non-zero position, “+”). If the control commands the valve to be shut, anyway, a stuck-closed valve would cause no deviation in the water flow. This is captured by the model fragment in Table 5, which actually, is the complete fault model. Table 4 and Table 5 show the models of the OK mode and the passing valve, respectively. The table expresses that this mode may coincide with the nominal behaviour for a certain range of opening commands, but deviate positively for smaller valve positions.

With respect to their use for diagnosis, tables 4 – 6 jointly with table 3 capture which tuples of temperature and water flow deviations are consistent with which behaviour modes. Note that this does not require that the deviations can be observed directly. They may also be **predicted** by the system model based on observations for a particular system health assignment.

Bear in mind that a qualitative representation of one mode doesn’t exclude that any other mode can be reached with the same combination of inputs/outputs.

### 5.1 Runtime Deviation Generation

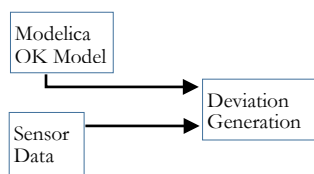


Figure 8 Generating Deviations

At runtime, the system will calculate deviations (Figure 8) by following the steps:

- Read each data vector corresponding to the sensor and actuator signals;
- Extract the exogenous variables including (external temperature, damper, and valve commands);
- Provide the exogenous variable values to the Modelica model of nominal behaviour, compare the values predicted by this model with the actual sensor data, and compute the deviations. In the current

Table 3. Relation on temperature deviations and water flow deviation

$\Delta m_{flow_w}$	$\Delta T_{al}$	$\Delta T_{aO}$
0	-	-
0	0	0
0	+	+
-	-	-
-	0	-
-	+	*
+	-	*
+	0	+
+	+	+

Table 4 Qualitative representation of the OK mode

Cmd	$\Delta m_{flow_w}$
0	0
+	0

Table 5 Qualitative representation of the stuck closed valve mode

Cmd	$\Delta m_{flow_w}$
0	0
+	-

Table 6 Qualitative representation of the passing valve mode

Cmd	$\Delta m_{flow_w}$
0	+
+	0
+	+

solution, this is simply done by using a threshold (which can be different for different variables).

For the example with the heating coil documented here, a threshold of 2°C was chosen in order to produce deviations in the domain of signs (+, -, 0). In future solutions, different orders of magnitudes of the deviations could be generated by the abstraction module, which can take arbitrary sets of interval boundaries as an input.

For the example with the heating coil, Table 7 shows both the sensor data and the predicted values, highlighting the temperature before and after the heating coil. Using the 2°C threshold, the inflow air temperature is determined as nominal, while the outflow air temperature is higher than expected. This triggers a diagnosis event.

### 5.2 Diagnosis Inference

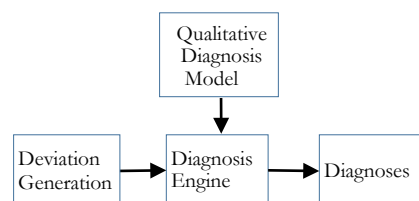


Figure 9 From deviations and qualitative model to diagnosis

The computed deviation pattern (with a zero deviation of exogenous variables -input temperature, and valve commands-) forms the input to the diagnosis runtime system (Figure 9). The deviation patterns will be checked for consistency with the possible models. In the trivial example restricted to one component presented in Table 7, the input/output temperature deviations (0, +) match with only one row in Table 3 that holds for all behaviour modes, which fixes  $mflow_w$  to be positively deviating. This positive deviation is consistent with the valve passing mode (Table 6), but neither with the OK mode, nor the stuck closed mode. Note, that this result can actually be concluded without information about the command to the valve.

What is illustrated here for a single component, is actually applied to the space of **plant** models covered by the system health assignments, which may yield alternative diagnosis hypotheses and also such that correspond to multiple component faults.

Table 7 Deviations between sensor data and model data

	$T_{ai}$ (°C)	$T_{ao}$ (°C)
Sensor Data	18.32	20.87
Model Prediction	18.44	18.44
<b>Resulting Deviation</b>	<b>0</b>	<b>+</b>

## 6 Discussion

In this paper a tool chain from model development to fault detection in air handling units has been presented and discussed with an illustrative example of a heating coil. The development tool of choice for the model was Modelica since it provides all the necessary tools to comply with model requirement for model-based fault detection as shown in section 3.1.

One of the main advantages of the model-based approach is the adaptability to different plants and to changes in the same plant. A brief description of the steps involved in adapting the qualitative model based diagnosis is presented below.

- **Structural changes:** These changes will have to be reproduced in the model, which would need to be compiled and recalibrated. The diagnosis model structure is a 1:1 mapping of the model and as such only minor adaptation is needed. However, if the change involves variables considered for diagnosis, the variable mapping between model and diagnosis framework has to be modified and tested with new data sets.
- **Parameter changes:** recalibration of the models is in principle the only requirement. In the case these parameter changes impact the accuracy of the

model, the tolerances of the diagnosis framework might have to be adjusted.

- **Sensor changes:** similar consideration to the case of structural changes should be taken in the case of adding new sensors or modifying position of existing ones. In the case that existing sensors are to be replaced with new ones with different precision, the steps described in the parameter changes are to be followed.
- **Changes in control:** plant model and diagnosis framework is, in principle, not affected by changes in the control strategy.

This adaptability makes model-based diagnosis a viable approach to fault detection and diagnosis in air handling units.

Taking into account that heating ventilation and air conditioning systems are rarely critical systems, the benefits of FDD in the build environment are more economic and environmental rather than being a safety issue and that hourly fault detection and diagnosis frequencies are more than acceptable in building applications; there is little scope for extending the models to include dynamic behaviour at the moment. Although in early stage, there exist scope for modelica models to become the de-facto standard in energy modelling of building components as shown by the recently established International Energy Agency Annex 60. Within this context, one of the key issues for model use during operation (e.g. Model-Based FDD, Model-Predictive Control, etc.) is the development of calibrated models that represent in a cost-effective manner the expected normal behaviour of the systems. Focused on air handling units' components, an approach to tack such problem, which can be automated, has been presented in this research paper

## 7 Future Work

Next steps in this research are:

- Development and testing of models for other components of HVAC systems;
- Improve the calibration methodology by developing an automatic calibration procedure that could be implemented underpinned by machine learning.
- Comparison of the qualitative model-based diagnosis approach with others such APAR rules or quantitative diagnosis
- Deployment and testing in a range of real units operating in normal environments.

## 8 Acknowledgements

This work was supported by the International Energy Research Centre and Enterprise Ireland under project n. CC-2011-4005B and by the Irish Research Council – D’Appolonia enterprise partnership scheme. Special thanks to Dominik O’Sullivan John McCarthy for their invaluable support and help in providing datasets for testing the developments presented in this research work.

## 9 Nomenclature

<i>eff</i>	effectiveness	[1]
<i>Q</i>	heat transfer	[W]
<i>C</i>	capacity flow	[W/K]
<i>T</i>	temperature	[°C]
<i>mflow</i>	mass flow rate	[kg/s]

### Subscripts and functions

<i>a</i>	air	<i>I</i>	input
<i>w</i>	water	<i>O</i>	output
$\min(\cdot, \cdot)$	smallest value between arguments		

American Society of Heating, Refrigerating and Air-conditioning Engineers, 2009.

- [6] M. Wetter, “Simulation Model: Finned Water-to-Air Coil without Condensation. LBNL-42355,” 1999.
- [7] P. Struss, “Models of Behavior Deviations in Model-based Systems,” in *European Conference on Artificial Intelligence*, pp. 883–887.
- [8] P. Struss and A. Fraracci, “Automated Model-Based FMEA of a Braking System,” *Fault Detect. Superv. Saf. ...*, no. Safeprocess, pp. 1–6, 2012.
- [1] S. Katipamula, P. Michael, and R. Brambley, “Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems— A Review, Part II,” *HVAC&R Res.*, vol. 11, no. 2, 2005.
- [2] R. Isermann, “Model-based fault-detection and diagnosis – status and applications,” *Annu. Rev. Control*, vol. 29, no. 1, pp. 71–85, Jan. 2005.
- [3] P. Struss, “Model-based problem solving,” in *Handbook of Knowledge Representation*, Elsevier, vol. 6526, no. 07, V. L. and B. P. F. van Harmelen, Ed. Elsevier B.V., 2008, pp. 395 – 465.
- [4] O. Dressler and P. Struss, “The consistency-based approach to automated diagnosis of devices,” in *Principles of Knowledge Representation*, G. Brewka, Ed. Stanford, CA, USA: Center for the Study of Language and Information, 1996, pp. 267 – 311.
- [5] ASHRAE, *ASHRAE Handbook: Fundamentals (SI edition)*. Atlanta, GA:

# Simulation for verification and validation of functional safety

Lars Mikelsons   Zhou Su  
Bosch Rexroth AG  
Rexrothstr. 3, 97816 Lohr am Main

## Abstract

Safety of machinery is the most critical issue in the design of mechatronic systems. The verification and validation procedure for functional safety of machinery is thoroughly discussed in ISO 13849-2. Following this procedure, the system behavior in case of a component failure has to be analyzed. Up to now this analysis bases on expert knowledge and real experiments. In this contribution a simulation based approach is presented. This approach has several advantages over the state-of-the-art. First, real experiments are more time consuming and costly than simulation. Moreover, according models can be used for further investigations like optimizing the sensor setup.

To enable failure simulation as a substitute of testing on real machinery for validation of functional safety, typical hydraulic failures are added to safety-related components of an in-house Modelica hydraulics library. This library is then used for the verification and validation of functional safety of a hydraulic test bench. Moreover, error propagation is considered.

*Keywords: functional safety; hydraulics; simulation; failure modeling*

## 1 Introduction

### 1.1 Motivation

Safety is of primary concern for all machine designers. The functional safety of a mechatronic system is assured by the correct execution of safety functions. Those parts of the complete system that are relevant for the execution of safety-functions are denoted as safety-related part of the control system (SRP/CS). ISO 13849 provides guidelines to assure safety of mechatronic systems. While ISO 13849-1 [1] concentrates on the design of the SRP/CS, ISO 13849-2 [2] focuses on the validation of functional safety. Thereby, the reliability of the execution of a safety function is evaluated by a discrete measure called performance level (PL). The determination of the PL of a safety

function requires the analysis of the system behavior in case of one or more component failures of the SRP/CS. The failures that need to be considered for that analysis are also standardized in ISO 13849. The PL is then used to verify that a mechatronic system is functional safe, by checking that the PL of the SRP/CS is greater or equal to the required performance level ( $PL_r$ ) of the system. Obviously, the  $PL_r$  has to be derived beforehand from a risk assessment.

It is obviously desirable, that the validation of a safety function can be done solely by analysis, using mainstream failure analysis techniques like Failure Mode and Effect Analysis (FMEA) [3] or Fault Tree Analysis (FTA). However, in most industrial applications, the SRP/CS of a safety function are too complex to analyze the system behavior by the engineers intuition. Hence, the result of these failure analysis techniques is seldom conclusive. Consequently, testing on the real system must be carried out in order to get a reliable result. For these tests usually a prototype of the SRP/CS has to be constructed, which is a time consuming and costly task. Furthermore, correct insertion of the desired component failure into the test setup is not only difficult, but can also damage the prototype, e.g. if the influence of contaminated oil is investigated. In some applications, testing on actual constructed systems might not even be possible, e.g. if the considered failure leads to a hazardous situation for the operator. This is the case in hydraulic applications, when the housing of a component breaks, because this leads to an eruption of the oil at high pressure. To overcome the same problems (costly and time consuming prototypes), years ago simulation was established as a development tool. The use of simulation for the verification and validation of requirements is visualized in the mechatronic V-Model (see figure 1). However, up to now the requirements were mostly functional requirements. To the author's knowledge there exists no methodology to use simulation for the verification and validation of functional safety with respect to ISO 13849. Consequently, there are no libraries available

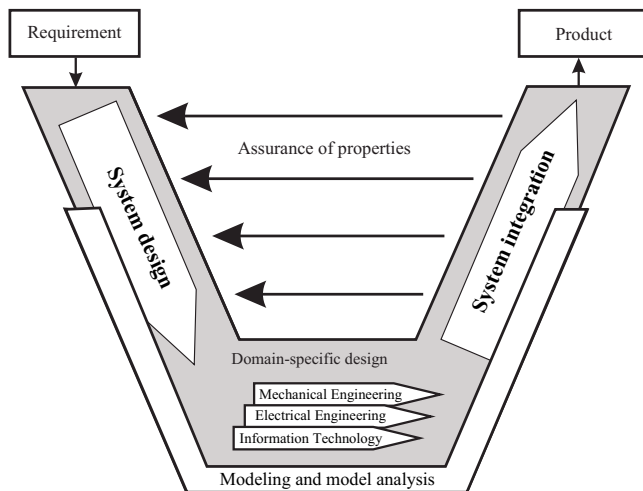


Figure 1: The mechatronic V-Model

that have a suitable level of detail, i.e. libraries including failure models. Obviously, it is desirable to model components including failures in such a way that

- the failures can be attached to a model without a failure (the original model),
- modification of the original model does not require modifications of the failures,
- the different failures can be easily exchanged,
- new failures can be added.

According to the first requirement physical models are needed. The remaining requirements aim for an object oriented model. Thus, Modelica seems to be appropriate to set up such models.

In this contribution an in-house hydraulics library is extended by components with failures, where the failures are modeled according to the requirements mentioned before. On the basis of that library it is shown how simulation can support and ease the design of a functional safe system. Moreover, an approach for error propagation is presented. A hydraulic test bench, which is used for testing hydraulic components like valves, is used as an application example. To summarize, the major benefits of the methodology presented here are:

- Optimization the typical work flow of design with respect to functional safety of mechatronic systems [4], i.e. replacing conventional analysis techniques and tests on prototypes by failure-simulation.

- Automated identification of the safety-critical failures in the SRP/CS.
- Investigation of error propagation.
- Future: Possibility to determine whether a failure can be detected by the sensor arrangements in the SRP/CS.

## 1.2 State-of-the-art

The concept of functional safety is derived from a functional system representation. Thus, most approaches for computer aided design of functional safe systems use functional models. A functional model is a block diagram of the system under consideration, where the blocks represent functions of the model and the connections represent a flow of energy, material or signals. Functional models can be generated at very early design stages, but suffer from the fact that they are rather rough, e.g. they do not include any dynamics.

Using these functional models, a Functional Failure Identification and Propagation framework is proposed for the analysis of functional failure propagation in [5]. However, this approach differs significantly from the method shown here and suffers from two major drawbacks. First, the level of detail of functional models is very low. Thus, the value of the gathered information is limited. Moreover, a special syntax and semantics are developed, so that existing models can not be used or upgraded for safety considerations. The same holds the methods described in [6] and [7] since the authors also use functional models. It is noteworthy, that the approach presented by Deng is originally intended for the verification of general requirements and can hence also be used for the verification of functional safety.

Most functional model-based analysis approaches are supported by failure analysis techniques like FMEA or FTA. An approach for combining both is presented in [8]. The coupling is done using the Systems Modeling language (SysML), a general-purpose modeling language. This method allows for automatic computation of a FMEA from a functional model, but suffers from the drawbacks described above. The use of ModelicaML [9] could be a promising future direction of the work presented here.

Although most simulation-based safety analysis methods use functional models, examples of physical models used for safety investigation can be found. A simple hydraulic system including a 4/3 directional valve, a motor-pump group and a cylinder is modeled with the help of bond graphs in [10] for



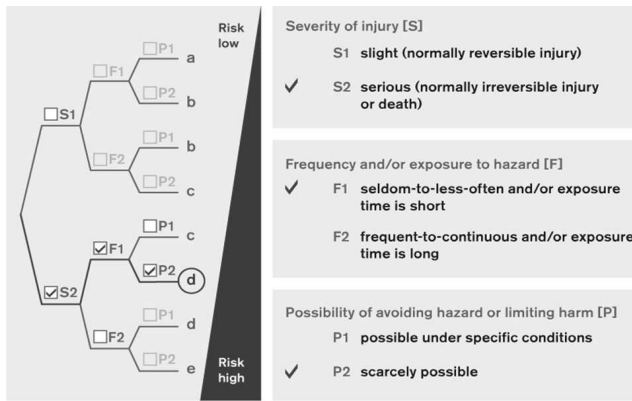


Figure 2: Determination of the required performance level using the risk graph

proactive fault diagnosis. However, without the help of powerful object-oriented equation-based modeling languages like Modelica, even modeling this simple hydraulic system is tedious and difficult. Therefore, the fault diagnosis is restricted to erroneous spool movement in the 4/3 directional valve. Moreover, this method does not satisfy the requirements stated in the motivation.

In the following section a brief introduction to functional safety is given. After that, a hydraulic library including models of failures is presented. Section 4 shows the application example mentioned before. The paper closes with a conclusion and an outlook.

## 2 Functional safety

Safety is the primary concern for every machine designer. ISO 12100 defines machine safety as:

*“the ability of a machine to perform its intended function(s) during its life cycle where risk has been adequately reduced.”*

If the machine safety depends on the correct functioning of a control system, the term *functional safety* is used. ISO 13849 contains guidelines for the design of a system with respect to functional safety. In [4] ten steps to reach the required performance level are presented. In the first step possible risks are identified and evaluated in a risk assessment. If required, measures to reduce the risks are chosen. These measures can be information for the use of the system, improved system design or safeguarding. If such a measure depends on the control system it is called a *safety function*. De-energizing of the system

in order to reach a safe state is a common safety function. In the second step the safety functions of the system are identified. In the third step the  $PL_r$  is determined for every safety function using the risk graph in figure 2. In the shown example the possibility of serious injuries (irreversible or death), that can happen only in short time span (e.g. 10min per hour) and are hard to avoid (e.g. fast moving machine) lead to a required performance limit  $PL_r = d$ . The  $PL_r$  quantifies the required reduction of the risk (see figure 3). Hence, after the third step the requirements for the SCRP/CS are known. Thus, in the fourth step the structure of the control system can be outlined. Following directive EN 954-1, control systems can be realized in the form of five categories (B, 1, 2, 3, 4) mapping the typical architectures, e.g. redundancy or additional shut-off paths. With each category only certain performance levels can be reached as can be seen in table 1. On the other hand, different choices for the category in order to reach a certain PL are possible. The fourth step is completed after the selection of an appropriate category. In the fifth step a functional model of the system is generated. That model is used in the sixth step order to analyze failures of the SCRP/CS. Therefore, a list of relevant failures is included in [2]. Basing on the functional model and the engineers expertise it is judged whether the failures from the failure list lead to a dangerous situation or the system remains in a safe state. Then, for each component the diagnostic coverage (DC) is determined as the ratio between failure rate of detected dangerous failures and the failure rate of total dangerous failures. Thereby, the failure rate is the inverse of the mean-time-to-failure (MTTF), which can usually requested for each component at the manufacturer (MTTF and  $MTTF_d$ ). The MTTF is the number of years at which approximately 63% would fail. Hence, the MTTF corresponds to a statistical, expected value and does not guarantee for a failure free time. The sixth step is completed after calculating the average diagnostic coverage for the complete SCRP/CS (consisting of N components) by

Category:	PL (possible)
B	a-b
1	b-c
2	a-d
3	a-e
4	e

Table 1: Reachable performance level

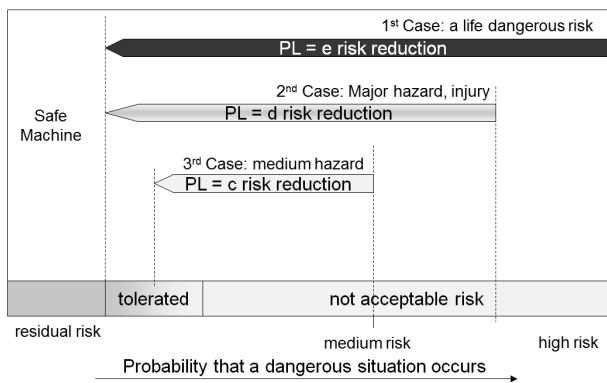


Figure 3: Principle of the risk reduction by the safety function

$$DC_{avg} = \frac{\frac{DC_1}{MTTF_{d1}} + \frac{DC_2}{MTTF_{d2}} + \dots + \frac{DC_N}{MTTF_{dN}}}{\frac{1}{MTTF_{d1}} + \frac{1}{MTTF_{d2}} + \dots + \frac{1}{MTTF_{dN}}}, \quad (1)$$

where  $MTTF_d$  denotes the mean time to a dangerous failure. In the seventh step the PL of the designed SCR/CS is determined. The PL depends on

- the category of the PLC,
- the reliability of the SCR/CS ( $MTTF_d$ ),
- the diagnostic coverage ( $DC_{avg}$ ).

There exist different approaches to compute the  $MTTF_d$  of the SCR/CS from the  $MTTF_d$  of the single components, e.g. Parts-Count-Procedure [4]. Using the  $MTTF_d$ , the category chosen in the fourth step and  $DC_{avg}$  determined in the sixth step, the PL can be read from a table given in [4]. Afterwards, the robustness of the PLC with respect to situations, that are not considered in the first steps, is analyzed in the eighth step. For example, in a redundant controller design all channels could fail due to violation of the maximum admissible operating temperature. In such cases adequate measures are taken. In the ninth step the software for the controller is developed with respect to state-of-the-art techniques and processes. In the last step the results from the previous steps are verified and validated. During the verification it is checked whether the required performance level has been reached. Otherwise the SCR/CS has to be improved, e.g. with components with a longer life cycle or a higher category. The plausibility of all the mentioned reliability parameters ( $MTTF_d$ , PL, category, DC), must be validated, either by analysis or with the help of testing/simulation. The complete validation procedure can be summarized as follows:

1. validation of safety functions
2. validation of performance level which includes:
  - validation of category specifications
  - validation of  $MTTF_d$  and  $DC_{avg}$
  - Validation of measures against systematic failure
  - validation of safety-related software
3. validation of combination and integration of all SR/CS

Theoretically, this verification and validation can be performed solely by analysis. However, due to the complexity of most control systems, testing or simulation must be carried out to support inconclusive failure analysis.

Besides the use of simulation for validation it is useful for some other tasks. In the sixth step dangerous failures are identified. Up to now this is done basing on a functional model and the engineers expertise. Due to typically big and complex systems this approach is time consuming and error prone. Hence, the use of simulation would not only speed up the design process, but also lead to more meaningful results.

### 3 Failure models in Modelica

The DC\_HydrauLib is a Modelica library for the simulation of hydraulic systems developed by Bosch Rexroth. It contains hydraulic components like pumps, cylinders and valves. In this contribution it is used to present a possible approach for failure modeling in Modelica. This approach respects the requirements stated in the introduction. Using the language features of Modelica these requirements can be satisfied in the following way: Each failure is implemented in a new model that extends from the nominal model. All these models are then collected in a wrapper model (via replaceable), which is denoted as the failure model. Modeling this way one failure model consists of multiple models including failures. During application the user can choose the failure, that should be investigated by a parameter. The failures are implemented according to the list from ISO 13849-2 containing typical hydraulic failures. This list is developed without consideration of modeling and simulation. Hence, each failure is described on the base of the actual component construction. Examples for failures of a switching valve are

- Change of switching times
- Non-Switching (sticking at the end or zero position) or incomplete switching (sticking at a random intermediate position)
- Spontaneous change of the initial switching position (without an input signal)
- Leakage
- Bursting of the valve housing or breakage of the moving component(s) as well as breakage/ fracture of the mounting or housing screws

However, the hydraulic library contains only models with a level of detail suited for system simulation. Thus, a translation into implementable descriptions fitting to the abstraction level of the existing hydraulic library is required first. While the translation for the first two failures is straight forward, different translations for the remaining failures are possible, e.g.

- Spontaneous change of the initial switching position (without an input signal): A white noise input signal replaces the control signal when the failure is triggered.
- Leakage: Internal leakage with user-specified leakage coefficient.
- Bursting of the valve housing or breakage of the moving component(s) as well as breakage/ fracture of the mounting or housing screws: External leakage with a (big) user-specified leakage coefficient.

A brief overview on a possible implementation of the failures is given below.

**Change of switching times** The switching behavior of a switching valve is modeled by a trapezoidal profile, that is parametrized by the switching times (on/off separately). In addition to the parameters and variables inherited from the base model, two new parameters,  $r_{on}$  and  $r_{off}$  are introduced, which are defined as

$$r_{on} = \frac{T'_{on}}{T_{on}} \quad (2)$$

$$r_{off} = \frac{T'_{off}}{T_{off}}, \quad (3)$$

where  $T$  and  $T'$  are the original and the erroneous switching time of the valve, respectively. In case of a failure the erroneous switching times are used in the calculation of the spool dynamics.

**Non-switching** This failure is split into two failures, non-switching and random sticking. Non-switching means that the valve cannot open or close upon the next opening or closing input signal. Thus, this failure does not take effect immediately at the moment the failure is triggered. However, at the moment the failure is triggered the current spool position is saved and from that moment on used for the flow rate calculation instead of the spool position calculated in the spool dynamics.

Random sticking, takes effect when the failure type is triggered, which means that the spool stays at the current position when the failure is activated at any random time. The sticking position can either be the two end positions or any intermediate position. This failure is very common in directional valves, when the spring holding the spool at the end position is broken. The implementation is very similar to the non-switching behavior.

**Spontaneous position change** This failure describes the situation when the valve becomes totally uncontrollable, most likely due to breakage of the spool. For this failure, a white noise generator producing filtered noise with a user-specified bandwidth is used as the input for the spool dynamics. Notice that this noise generator is not really random. Only different seeds generate different noise output.

**Leakage** Leakage is an unavoidable problem in the construction of hydraulic valves. Leakage between two hydraulic ports is modeled as a volume flow rate proportional to the pressure difference  $\Delta p$  between the two ports

$$q = c \cdot \Delta p \quad (4)$$

with  $c$  as the leakage coefficient. Here, one has to distinguish between internal and external leakage. Internal leakage takes place between the two hydraulic ports of a the valve. In the example of a switching valve with two ports (A and B), oil might flow from port\_A to port\_B, even when the valve is fully closed. External leakage is leakage to the environment (modeled as a tank). This model can also be used to simulate the breaking of the housing or similar failures.

Looking at the failures it is clear that there should be some kind of triggering mechanism. In this work two different triggering mechanisms have been implemented. In the first case the failure is

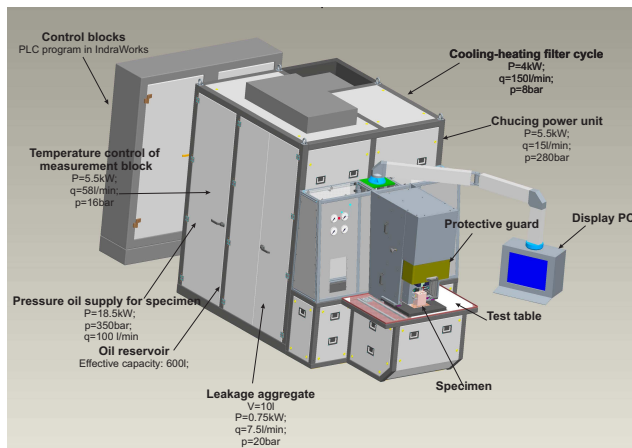


Figure 4: Hydraulic test bench

triggered at a user specified time. In the second case the failure is triggered by the violation of the working conditions of the corresponding component, e.g. pressure limitation, temperature range or fluid contamination. In this contribution only one working condition (pressure limits) is implemented. According to the specifications of the DC\_HydrauLib, both ambient and fluid temperature are constant during one simulation run, so exceeding the temperature range cannot be modeled using that library (until now). Similarly, fluid quality is also a fixed property of a selected oil type, thus contamination of oil can also not be modeled. Though, the user can specify a maximal operating pressure. If this maximal pressure is exceeded the chosen failure is triggered.

## 4 Application Example

In this section a test bench for hydraulic valves is used to present possibilities of the usage of failure simulation during the design phase of a hydraulic system. A typical safety function of this test bench (emergency stop initiated by user), is thoroughly investigated by simulation on the SRP/CS that executes this safety function.

### 4.1 Test bench

The hydraulic test bench (Figure 4) is designed for the testing of directional valves. The test bench is manufactured by Bosch Rexroth and applied in the production line of a Rexroth plant. Typical valve characteristics of the test item (specimen in figure 4) like leakage, characteristics curves and switching times can be tested. To perform all tests required for a test item,

the test bench must be able to provide various pressure and volume flow rates. During the productive period of the test bench, a trained operator stands in front of the test table to mount the test item. A test can only be started when the safety door (protective safeguard in figure 4) is open. On the other hand this door can only be lowered when the test item is correctly mounted. The test results are automatically recorded by the control devices of the test bench. Thereby, the test results can be severely impaired by a malfunction of the test bench, especially by component failures in the motor-pump group and the test table. For example, internal leakage inside the test bench may lead to test results, that indicate too big leakage of the test item. Moreover, if one of the safety functions, which are measures against risks, cannot be carried out due to internal component failures, the consequence can be even more disastrous. Thus, each time before the test bench is started a checking routine has to be performed. To reduce this non-productive period, and at the same time ensure the correct functioning of safety functions like emergency stop and safe door locking, is one of the most challenging issues in the designing of hydraulic test benches.

### 4.2 Circuit example

The safety function to be investigated for the hydraulic test bench (emergency stop initiated by user), is activated by pressing the emergency button on the control panel, and executed by de-energizing all engaged valves. Notice that in the execution of this safety function only the relevant actuators, namely valves engaged in cutting off the pressure supply, are de-energized. The energy supplier (the electrical motor within the motor-pump group) is still working, because it might be employed in other crucial functions of the test bench.

Two functional blocks exist between the oil pressure supply and the test table, where the test item is mounted. The block directly below the base frame of test table is the measurement block, which contains mainly sensors for measuring and valves for channel selection. This measurement block is designed to carry out all important tests on the test item. Notwithstanding its importance in the correct functioning of the test bench, it is not safety-related, since it does not execute a safety function. Moreover, malfunction of this measurement block only results in the non-execution of the designed tests on the test item. No danger is caused by the loss or degradation of this function. So the measurement block is out of the investigation scope of this

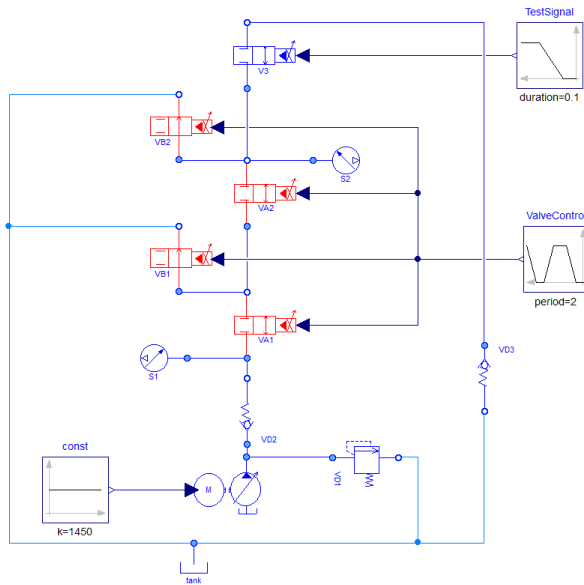


Figure 5: Simplified circuit diagram

contribution.

The other block is directly connected to the pressure supply. This block is designed to cut off the oil supply from the pump as fast as possible by de-energizing all engaged valves upon emergency. So this functional block is the active part to execute the safety function under consideration, and thus in the focus of the investigation in this contribution. The original system consists of several identical channels that can be connected to the test item. However, for the following investigations only one channel is considered for the reason of simplicity. Some additional simplifications lead to the circuit diagram in figure 5. Here, the red components (valve VA1, VB1, VA2 and VB2) are failure models. An explanation of the essential components is given in the following:

- VA1 and VA2: These 2/2 directional valves control the pressure supply of the test table. The valves have a switching time of 10ms and an internal induction sensor for spool monitoring (high DC value).
- VB1 and VB2: These 2/2 directional valves are opened in order to let the remaining oil between two valves flow back to the tank. The valves have a switching time of 10ms.
- VD1: This pressure relief valve limits the pressure in the circuit to 250bar.
- V3: This 2/2 switching valve imitates a test item.

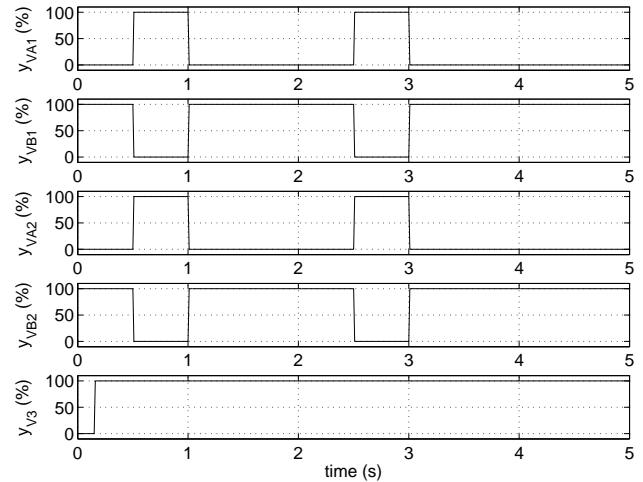


Figure 6: Spool position of the valves in the reference simulation

### 4.3 Failure simulation

Before performing failure simulations, a failure-free simulation is first carried out as a reference behavior. During this reference simulation the safety function is triggered twice within five seconds. Thereby, the motor-pump group provides a constant volume flow rate of 40 l/min. Figure 6 shows the spool position  $y$  of all controlled valves (VA1, VB1, VA2, VB2 and V3). The safety function is triggered at 1s and 3s, respectively. Thus, at these times the valves VA1 and VA2 are de-energized (closed). Moreover, VB1 and VB2 also de-energized (opened) and thus let the residue oil flow back to the tank. Valve V3, which represents the test item, is controlled by a separate testing signal, and is not engaged in the execution of the safety function. It opens at 0.15 s and stays completely open throughout the whole simulation. The volume flow rate into the test item  $q_{V3}$  is the output variable since this variable is a measure for the danger of extruding oil. The volume flow rate into each major (VA1, VA2, VB1, VB2 and V3) is shown in figure 7. Valve VB1 and VB2 are auxiliary valves designed to let out residue flow when the main channel is suddenly cut off. Hence, the flow rate trough these valves is much smaller than the flow rate through VA1 and VA2.

The results of the reference simulation is used as the reference for all failure simulations performed in the following.

#### 4.3.1 Time triggered failures

In the first step described in Sec. 2 a risk assessment is performed. Here, simulation with failures injected

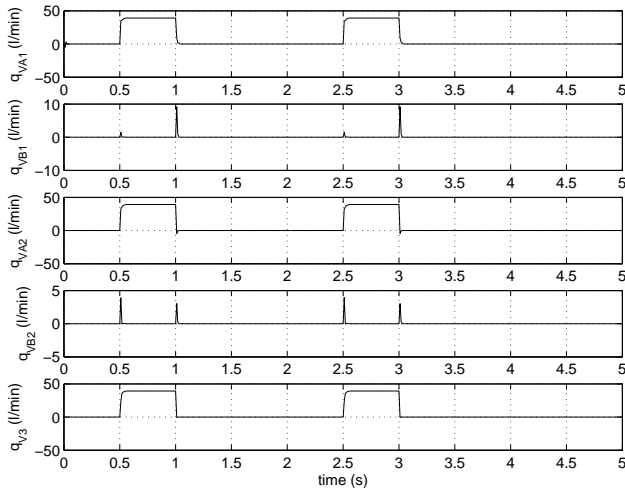


Figure 7: Volume flow rate at port\_A of all controlled valves in the reference simulation

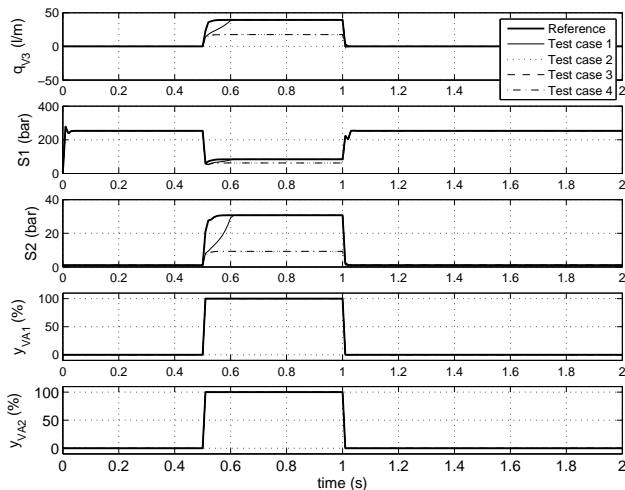


Figure 8: Comparison of four failure simulations with the reference simulation

into only one of the safety-related components (VA1, VA2, VB1 and VB2) can be used in order to get better estimates of the consequences of component failures. In the sixth step it judged whether a failure leads to dangerous situation or not. Clearly, also here simulation can be used. However, here the use of simulation in the tenth step is shown, i.e. how to use simulation for validation. Therefore, the category of the SRP/CS is validated in the following. The chosen category for the system at hand is three ( $PL_r=d$ ). According to ISO 13849-2 this requires that a single failure does not lead to the loss of the safety function. Therefore, a Dymola script is used to simulate possible single failures. In figure 8 some typical results are presented. The trajec-

tories under investigation are the system output  $q_{V3}$ , which is the volume flow rate into the test item valve V3, and all measurable states, which are the displays of the two pressure sensors S1, S2 and the two internal sensors monitoring the spool position for valve VA1 and VA2. Sensor S1 is placed near the pump, and is used to monitor the pressure source of the whole system. Sensor S2 is placed at the inlet port of the test item valve V3, and is used to monitor the pressure provided to the test item.

It can be easily seen that the trajectories of  $q_{V3}$  and the two pressure sensor displays in the four test cases in figure 8 differ from the reference trajectories around 0.5s (valve VA1 and VA2 are opened). However, after the first triggering of the safety function at 1s,  $q_{V3}$  becomes almost identical to the reference trajectory for each test case. The same holds for all test cases not shown here. Thus, the safety function is successfully executed in all the four test cases and the safety function is not lost in case of a single failure.

Note that the validation of the category can thus be performed only based on simulation, which saves time and money (especially for more complex systems). Clearly, during the risk assessment also combinations of failures can be simulated.

#### 4.4 Error propagation

For the previous simulations, only time-triggered failures are considered. Another option for the failure triggering mechanism, in which a component failure is activated when a safety principle is violated, is considered in this section, for the investigation of error propagation.

Therefore, the failure-free pressure relief valve VD1 in figure 5, which is used for pressure limitation of the whole system, is replaced by an equivalent failure model. Note that the failure lists for pressure valves differs from the failure list for directional valves. The only investigated failure in this contribution is non-opening, which means that the valve cannot open completely (maximal 1%).

In the test case for error propagation the failure of the valve VD1 is triggered at 1s. Additionally, VA1 becomes uncontrollable (spontaneous spool movement), if the pressure at the inlet port exceeds the pressure limit of 400bar. The other valves (VA2, VB1 and VB2) exhibit a big external leakage (corresponds to the breaking of the housing), if the pressure limit (same as for VA1) is exceeded. Once again, a simulation of five seconds with two requests on the safety function is performed (denoted as Testcase 1 in the fig-

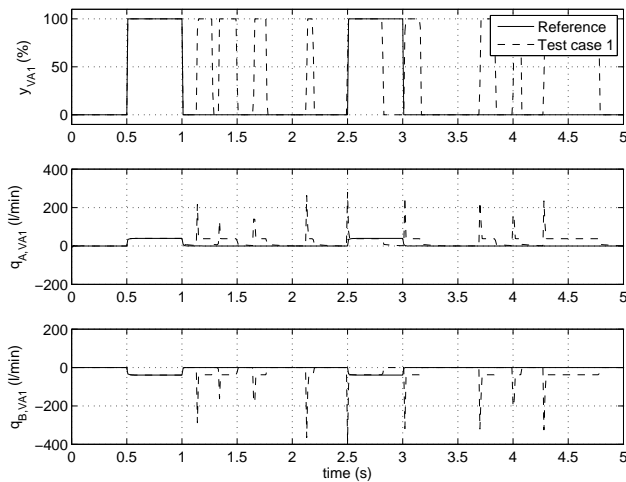


Figure 9: Results of the valve VA1 compared with the reference simulation

ures). In figure 9 the simulation results for valve VA1 are shown. It can be seen that the valve VA1 breaks down shortly (at 1.01s) after the failure of valve VD1 is triggered, because the pressure at the input port gets bigger than the pressure limit. From that time on the spool moves uncontrolled, which results in the presented volume flow. The valves VA2 and VB1 break down simultaneously shortly after the break-down of valve VA1 at 1.13s. This is, because the inlet ports of these two valves are connected to the same hydraulic port. The break-down of the valves VA2 and VB1 is modeled as big external leakage and hence these valves nearly act a tank for the rest of the simulation. That behavior explains the simulation results of the valve V3 (shown in figure 10). It can be seen that the volume flow after the failure of VA2 is much smaller than in the reference simulation. Note that the valve VB2 works properly, which indicates that the pressure at the end connector to the test item does not exceeds the maximum allowed pressure. This can also be confirmed by a look at the pressure sensor S2, which is even lower as in the reference simulation. This behavior occurs, because the valves VB1 and VA2, which are nearer to the pressure supply, break down earlier (and exhibit a big external leakage as explained before).

This example shows that it is possible to perform a model based estimation of the consequences of component failures, where even error propagation is considered.

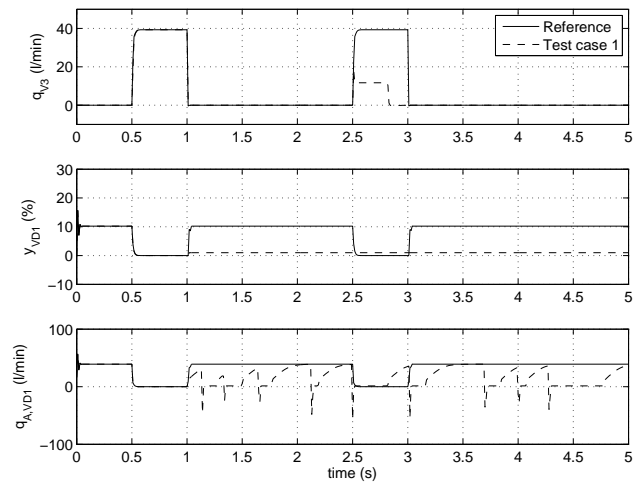


Figure 10: Results of the valves V3 and VD1 compared with the reference simulation

## 5 Summary

In this contribution an approach for the use of simulation for the verification and validation of functional safety is presented. Therefore, a hydraulics library is extended by failure models. Thereby, the failure models depend on the original model in such a way, that the requirements stated in the introduction are satisfied. One possible use of these failure models is a risk assessment. The simulation of single failures lead to insight in the consequences of component failures. Here, also error propagation can be considered. Furthermore, the failure models can be used for the verification and validation step, e.g. the verification of the category as shown before.

In the future, optimization will be used for the identification of the worst case combination of failures. The results can either be used for the risk assessment or the identification of critical components. Moreover, the approach can be used in order to identify an optimal sensor setup. Therefore, on the one hand better sensor models have to be implemented and on the other hand algorithms for failure identification are required.

## References

- [1] ISO 13849-1: Safety of machinery-safety-related parts of control systems - Part 1: General principles for design. International Organization for Standardization (ISO), 2006.
- [2] ISO 13849-2: Safety of machinery-safety-related parts of control systems - Part 2: Validation.

- International Organization for Standardization (ISO), 2010.
- [3] Bertsche B. Reliability in Automotive and Mechanical Engineering. VDI-Buch, Springer-Verlag Berlin Heidelberg, 2008.
- [4] Barg J., Eisenhut-Fuchsberger F., Orth A. 10 steps to performance level - Handbook for the implementation of functional safety according to ISO 13849, 2012, Bosch Rexroth AG
- [5] Sierla S., Tumer I., Papakonstantinou N., Koskinen K., Jensen D. Early integration of safety to the mechatronic system design process by the functional failure identification and propagation framework. In: Mechatronics, Volume 22, 2012.
- [6] Belmonte F., Schön W., Heurley L., Capel R. Interdisciplinary safety analysis of complex socio-technological systems based on the functional resonance accident model: An application to railway traffic supervision. In: Reliability Engineering and System Safety, Volume 96, 2010.
- [7] Deng A., Britton G., Tor S. Constraint-based functional design verification for conceptual design. In: Computer-Aided Design, Volume 32, 2000.
- [8] David P., Idasiak V., Kratz F. Reliability study of complex physical systems using SysML. In: Reliability Engineering and System Safety, Volume 95, 2009.
- [9] Schamai W., Fritzson P., Paredis, C., Pop A. Towards unified system modeling and simulation with ModelicaML: modeling of executable behavior using graphical notations, In: Proceedings 7th Modelica Conference, Como, Italy, 2009
- [10] Athanasatos P., Costopoulos T. Proactive fault finding in a 4/3-way direction control valve of a high pressure hydraulic system using the bond graph method with digital simulation, In: Mechanism and Machine Theory, Volume 50, 2012



# The Foundation of the DLR RailwayDynamics Library: the Wheel-Rail-Contact

Andreas Heckmann<sup>◊</sup>, Alexander Keck<sup>◊</sup>, Ingo Kaiser<sup>◊</sup>, Bernhard Kurzeck<sup>◊</sup>  
<sup>◊</sup>German Aerospace Center (DLR), Institute of System Dynamics and Control  
Oberpfaffenhofen, 82234 Wessling, Germany  
[andreas.heckmann@dlr.de](mailto:andreas.heckmann@dlr.de)

## Abstract

The formulation of the wheel-rail contact is a crucial issue in simulations considering the running dynamics of railway vehicles. Therefore a modeling environment that is dedicated to railway vehicle dynamics such as the new DLR RailwayDynamics Library relies on an efficient representation of the kinematics and forces or torques, respectively, that appear at the wheel-rail interface. A number of different formulations have been developed since the underlying rolling contact problem was firstly discussed in literature in 1876. The paper overviews these wheel-rail contact formulations and then presents the implemented variants in detail. The DLR RailwayDynamics Library is used to model and simulate the behavior of an experimental scaled M 1:5 running gear operating on the DLR roller rig. The simulation results are compared and validated with measurements. *Keywords: railway vehicles; wheel-rail contact; vehicle dynamics; multibody simulation*

## 1 Literature Review

### 1.1 Introduction

The wheel-rail interface is a constitutional element of railway vehicles. *Knothe* et al. [1] tell its three fundamental tasks each associated to a specific contact force component: load-bearing to the vertical force, guiding to the lateral force and traction to the longitudinal force. Although this appears to be very similar to the tire-road interface for automotive vehicles, the wheel-rail interface differs fundamentally due to wheel-rail geometry and the different material behavior of both contact partners which are made of steel.

The contact between wheel and rail in normal

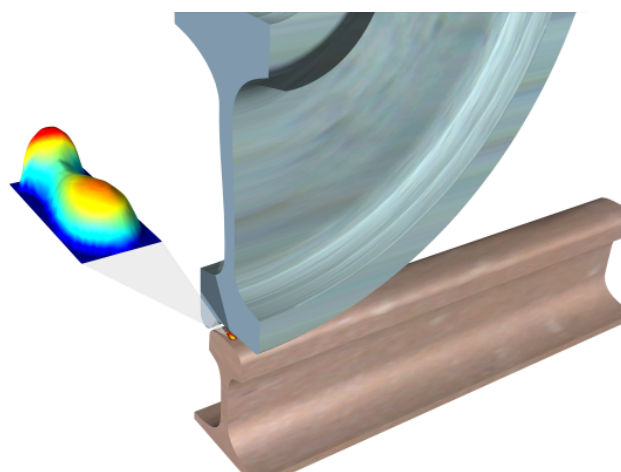


Figure 1: Exemplary contact patch between wheel and rail and a qualitative surface plot of the associated normal stress distribution

direction is very stiff. The deformations of the contact partners add up to 1/10 or 2/10 mm, the contact area has an approximate size of 1 to 2 cm<sup>2</sup> although the usual transmitted vertical loads are very high, i.e. in the order of magnitude of 10 tons. Fig. 1 gives an impression of the size of the contact patch and the normal stresses here presented with a maximum of roughly 700 N/mm<sup>2</sup>.

In the tangential direction the contact behavior is ruled by friction. Therefore it depends on the normal contact conditions with friction coefficients between 0.1 and 0.4 and exhibits non-linear behavior and saturation.

This general specification already exposes the complexity of the wheel-rail contact problem which is also demonstrated by Fig. 2. There, the contours visualize the normal stress distribution as it is already given by the surface plot in Fig. 1. The arrows represent the tangential stress vectors and the circles indicate the points where the wheel surface slips along the rail surface since adhesion

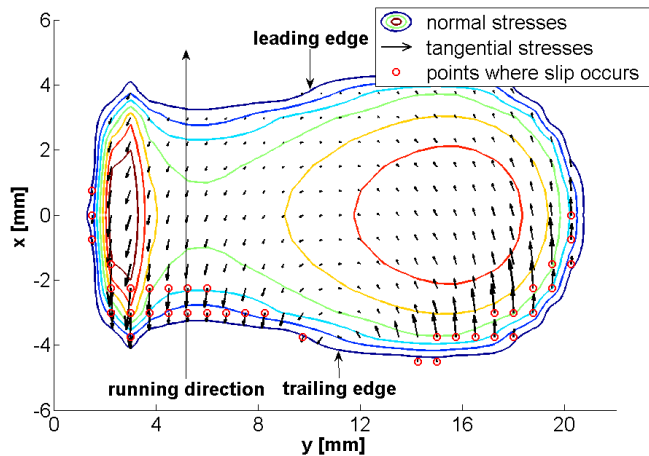


Figure 2: Exemplary contact stresses and the associated slip region evaluated according to *Kalker's* non-elliptical rolling contact theory [2]

is depleted locally.

The fundamental normal and tangential contact properties motivate the use of conical wheel profiles so that the normal contact does not only carry vertical loads but as well contributes to the lateral guiding task. The additional consideration of undesired profile changes due to wear led to the evolution of specific wheel and rail profiles as they are shown in Fig. 3, see [3] for a more elaborate discussion on the significance of the profile geometry. It is obvious that such profile design introduces another complexity into the formulation of a wheel-rail contact model due to the geometry. For example, the shape of the outer stress field boundary in Fig. 2, usually denoted as non-elliptical, is a particular result introduced by the non-linear profile geometry.

## 1.2 Basic Modeling Issues

The goal of the modeling presented in this paper is to provide the capability of simulating the running dynamics of railway vehicles. Hence, it is important that the resultant forces that dominate the motion of the railway vehicle and are to be computed very often during one simulation job, are evaluated in reasonable accuracy and with low computational burden. However, it is not intended to give detailed and high accuracy information on the contact stress distribution as shown Fig. 2 and as it is required in order to examine e.g. rolling contact fatigue.

It is therefore a usual approach to review the

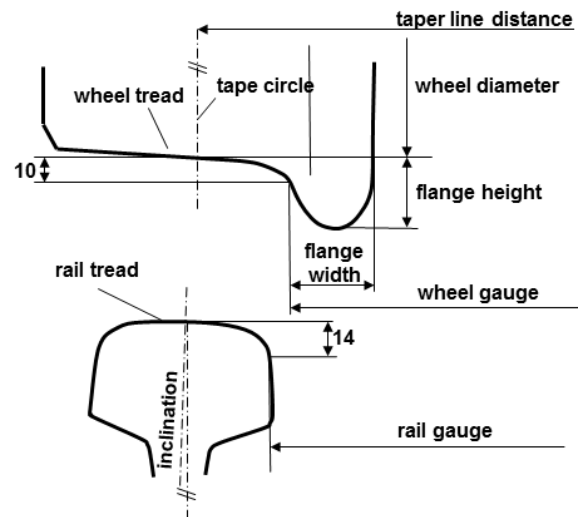


Figure 3: Terminology and characteristics of wheel and rail profile

wheel-rail contact problem looking for reasonable assumptions that simplify the modeling and reduce the computational effort. Tab. 1 gives an overview on widespread assumptions which may be applied according to the specific analysis goals at hand.

The separation assumption for example allows for introducing a virtual contact point where the resultant forces and torques acting on the wheel are assumed to be attached in order to simulate running dynamics. However the magnitude of the forces and torques are evaluated considering the contact problem separately.

Presuming identical materials for wheel and rail, the separation assumption together with the half-space approximation makes it possible to evaluate the normal contact first and independently and then derive the tangential stress quantities on top of it [1, Sec. 3.3.2].

Combining these two main simplification ideas with the discussion in Sec. 1.1 facilitates the formulation of the wheel-rail rolling contact problem by subdividing it into the following three subtasks that may be solved subsequently: the geometrical problem, the normal contact problem and the tangential contact problem.

A basic modeling aspect that is related to the multibody representation of the wheel-rail contact is the distinction drawn between rigid and elastic contact. In multibody theory the latter is equivalent to the description of the normal wheel-rail contact as a force law, while the ideal rigid contact leads to the concept of a kinematical constraint, so

Separation assumption	The deformations of the contacting bodies do not influence their overall motion so that wheel and rail may be assumed to be rigid bodies. Structural and contact mechanics may be considered independently from each other.
Linearity	The deformation fields are geometrically linear.
Half-space assumption	The contact area is very small compared to the characteristic dimensions of the related bodies in contact.
Ideal material	The material is linear-elastic, homogeneous and isotropic.
Identical material	Wheel and rail consist of identical materials.
Steady state assumption	The running velocity is small compared to the wave propagation within the material, so that quasi-static conditions are valid with respect to the contact problem.
Hertz assumption	The geometry of the contacting surfaces may be approximated as elliptic paraboloids so that the contact patch turns out to be a plain ellipse.
Dry Coloumb friction	Only dry friction is considered. The friction coefficient is constant and valid for adhesion as well as for sliding.

Table 1: Overview on assumptions frequently exploited for wheel-rail contact modeling, cp. [1, Ch. 3]

that the equations of motion form a differential-algebraic system [4]. The rigid contact requires a sophisticated preprocessing to guarantee the differentiability of the constraint equation and avoid artificial contact point jumps [5]. However, it is assumed to need less computational effort for time integration compared to the elastic contact that introduces very high stiffnesses into the equations of motion, but provides a more general applicability [6].

In the discussion so far it is assumed that the contact between wheel and rail forms one continuous contact area that may be idealized by one single contact point, which in fact is the dominant standard case. However specific configurations such as switch crossing or light urban and metro railway vehicles in sharp curves exist where multiple, non-connected contact areas between wheel and rail surface occur. These configurations require the consideration of multiple contact points, see e.g. [7] or [4].

### 1.3 The Normal Contact Problem

The basis for the highly accurate non-elliptical contact description e.g. shown in Fig. 2 has been set by *Kalker* [8] who implemented the program *Contact* that became a reference for railway contact problems. *Contact* fully accounts for the profile geometry of wheel and rail so that the simplification of the Hertz assumption from Tab. 1 is not employed. However, the half-space assumption is exploited to evaluate the stress and deforma-

tion field numerically. *Contact* is mainly applied for verification purposes in offline calculations. In addition, the same methodology has been used for detailed research about the influence of structural dynamics of wheel and rail on the vehicle-track interaction [2]. However for industrial applications, the accurate consideration of the non-elliptical contact requires too many computational resources, see e.g. [9] for optional approximations.

A very frequently used normal contact model in railway dynamics analysis is the *elliptical contact*. Here, the deviation of the contacting surfaces of wheel and rail from the ideal ellipsoidal shape is neglected and the contact stresses and deformations may be evaluated analytically according to the Hertz theory, see e.g. [10, Sec. 4.II.A].

For the sake of completeness the *Finite Element Method* shall be mentioned as a very general way to evaluate contact problems that does not rely on any of the mentioned simplifications in Tab. 1. Effects such as surface hardening, material flows or damage mechanism can be taken into account, see e.g. [11].

### 1.4 The Tangential Contact Problem

As soon as a relative motion, the so-called slip, between the contact point on the wheel and its counterpart on the rail occurs, tangential forces are transmitted. The program *Contact* is a widespread accepted reference to solve as well this tangential contact problem.

However in order to facilitate vehicle simu-

lations, *Kalker* proposed a simplified theory of rolling contact based on discretized ellipses and provided the *Fastsim* [12] algorithm that nevertheless takes traction and saturation into account.

For vanishing slip, if e.g. only very small traction forces are given, the linearized theory of *Kalker* [8, Sec. 2.2.2] is valid. Here, the tangential forces are a linear function of the slip.

A different class of tangential contact models try to get along with purely analytical considerations, as it is already done for the normal contact using the Hertz solution. However, these contact models require an assumption concerning the shape of the stick and slip region within the contact area. The basic idea originates from *Carter* [13], who showed that the tangential stress distribution of a cylindrical body rolling on a plane may be presented by two nested ellipses. The extension to three dimensions however relies on approximations that were proposed e.g. by *Vermeulen and Johnson* or by *Shen, Hedrick and Elkins* [13].

Following the same basic concept *Polach* [14] published a very efficient tangential contact solution in 1992 that coincides with *Kalker's* linear theory in the case of vanishing slip. In addition an extension is proposed for applications on the adhesion limit in which high traction forces are involved [15]. Besides the *Fastsim* algorithm the *Polach* model is wide-spread for multibody railway vehicle simulations today, see [16] for an assessment of various approaches.

## 1.5 Review Conclusions

The Modelica implementation of the wheel-rail contact is in particular supposed to support the development of new railway vehicle control concepts that are on the agenda of the DLR internal project Next Generation Train (NGT) [17]. For this purpose accuracy and computational effort have to be compromised. Therefore it has been decided to implement a rigid elliptical single point contact with tangential force law according to *Polach*.

## 2 Theory

### 2.1 Profile Geometry and Contact

In order to formulate the rigid contact, we look for an implicit constraint equation that defines the vertical wheel displacement  $z_w$  as a function of the lateral displacement  $y_w$ , the yaw angle  $\psi$  and the

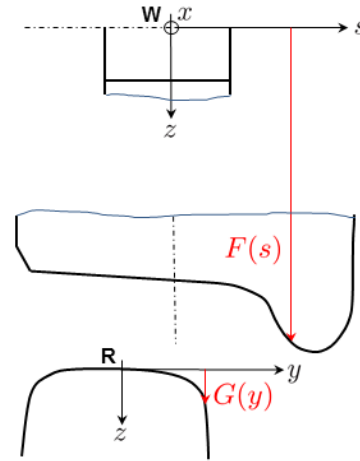


Figure 4: Parametrization of the profile geometry.

roll angle  $\varphi$  of the wheel [5], i.e.

$$z_w + f(y_w, \varphi, \psi) = 0, \quad (1)$$

which is two times continuously differentiable [18].  $z_w$  and  $y_w$  are resolved with respect to the rail coordinate system R in Fig. 4, where the parametrization of the wheel profile  $F = F(s)$  and the rail profile  $G = G(y)$  is visualized as well.

The wheel surface  $S_w$  may be given in cylindrical coordinates of the wheel, i.e.  $S_w : (F(s), \tau, s)^T$ , or resolved with respect to R:

$$S_w : \mathbf{c}_S = \begin{pmatrix} 0 \\ y_w \\ z_w \end{pmatrix} + A(\varphi, \psi) \begin{pmatrix} F(s) \sin \tau \\ s \\ F(s) \cos \tau \end{pmatrix}, \quad (2)$$

where  $A(\varphi, \psi)$  represents the rotations around the  $x$ - and the  $z$ -axis of the wheel.

Those points on  $S_w$ , whose normal vectors to the surface are parallel to the  $(y, z)$ -plane of the rail define the curve  $C_w$  that is the projection of the wheel contour in the  $(y, z)$ -plane of the rail [19, Sec. 2.2]:

$$C_w : \mathbf{c}_C = \left\{ \begin{aligned} \mathbf{c}_S : \sin \tau(s) &= -\tan \psi F(s), \\ \cos \tau(s) &= \sqrt{1 - \sin^2 \tau(s)} \end{aligned} \right\}. \quad (3)$$

We now specify the curve  $C_E$  to be independent from the vertical position of the wheel  $z_w$

$$C_E : \mathbf{c}_E = (x_E, y_E, z_E)^T := \mathbf{c}_C - (0, 0, z_w)^T, \quad (4)$$

which can be exploited to relate the wheel contour  $z_E = z_E(s; y_w, \varphi, \psi)$  to the associated rail profile

$G(y_E) = G(s; y_w, \varphi, \psi)$  as a function of the parameter  $s$ ,  $\underline{s} \leq s \leq \bar{s}$ .

Wheel and rail are in contact iff

$$z_w + \max_{\underline{s} \leq s \leq \bar{s}} \underbrace{(z_E - G)}_{\Delta(s; y_w, \varphi, \psi)} = 0. \quad (5)$$

The function  $\Delta(s; y_w, \varphi, \psi)$  in (5) is called the height function [5]. The value  $s^*$  that is assigned to the global maximum of  $\Delta$  specifies the contact point.

The direct application of (5) as constraint equation in multibody simulation for standard wheel-rail profiles such as S1002 and UIC60 does not make sense, since these profiles expose abrupt curvature changes and promote artificial jumps of the normal contact forces [4]. Therefore *Arnold et al.* [5] propose to use a regularization parameter  $\alpha > 0$  in the range  $10^{-5} \dots 5 \cdot 10^{-5}$  in the following way:

$$z_w + \text{smax}_s^{(\alpha)} \Delta(s, y_w, \varphi, \psi) = 0, \quad (6)$$

$$\text{smax}_s^{(\alpha)} \Delta := \alpha \ln \frac{\int_{\underline{s}}^{\bar{s}} \exp\left(\frac{\Delta(s, y_w, \varphi, \psi)}{\alpha}\right) ds}{\int_{\underline{s}}^{\bar{s}} ds}.$$

For small values of  $\alpha$  it can be shown that  $\text{smax}_s^{(\alpha)} \Delta \leq \max_s \Delta$ , i.e. (6) yields values of  $z_w$  that represent a small penetration  $\delta_{rw}$  of the wheel and rail bodies. The proposed values of  $\alpha$  are chosen in such a way that  $\delta_{rw}$  corresponds to the physical deformations of the contact partners, which could be evaluated e.g. according to the Hertzian theory. Therefore (6) is called the quasi-elastic contact model by *Arnold et al.*

The listing below shows that the geometrical problem is tackled by a Modelica function that takes  $y_w$ ,  $\varphi$ ,  $\psi$  and  $\alpha$  as inputs and mainly returns  $z_w$  in addition to quantities that are necessary for the tangential contact evaluation:

```
function findQuasiElasticContact
  "finds contact points on wheel and rail"
  import Modelica.Constants.pi;
  import SI = Modelica.SIunits;
  input SI.Position y_w
    "lateral displacement of wheel";
  input SI.Angle phi    "roll angle [rad]";
  input SI.Angle psi    "yaw angle [rad]";
  input SI.Radius r0    "nom. wheel radius";
  input Real alpha     "smoothing parameter";
  output SI.Position z_w "vertical
    displacement of wheel center point";
  output SI.Position s0
    "lateral contact coordinate on wheel";
  output SI.Position v0
```

```
"lateral contact coordinate on rail";
output Real rho_x(final unit="m-1")
"principal curvature at contact point
  in the plane normal to x-axis";
output Real rho_y(final unit="m-1")
"principal curvature at contact point
  in the plane normal to y-axis";
output SI.Angle delta "contact angle";
```

The quantity  $s$  is an internal vector variable of this function which discretizes the wheel profile, e.g.  $s = \{-0.05, -0.0499, \dots, 0.05\}$ .

## 2.2 Kinematics

Consider the coordinate system in Fig. 5 in order to resolve the vectorial quantities in what follows. The  $e_z$ -vector is normal to the wheel and the rail surface in the contact point C,  $e_x$  is perpendicular to the wheel axis and heads in running direction.  $v_0$  is that component of the absolute velocity of the wheel center point  $v_w$  that points in  $e_x$ -direction.  $\omega_w$  is the absolute angular velocity of the wheel that includes yaw, roll and the overturning motion  $\omega_0$  of the wheel.

The sliding velocity  $v_s$  [20, Sec.2.6.2] in C follows from

$$v_s = v_w + \omega_w \times r_C \quad (7)$$

and is used together with  $\omega_w$  to compose the slip vector  $\nu$  with the longitudinal slip  $\nu_x$ , the lateral

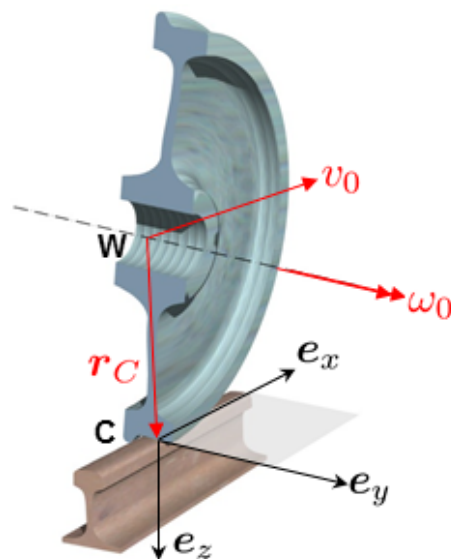


Figure 5: Coordinate system associated to the plane through contact point C.

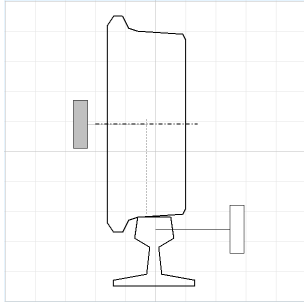


Figure 6: Icon layer of the contact model.

slip  $\nu_y$  and the spin  $\phi_z$ :

$$\boldsymbol{\nu} := \begin{pmatrix} \nu_x \\ \nu_y \\ \phi_z \end{pmatrix} = \frac{1}{v_0} \begin{pmatrix} v_{sx} \\ v_{sy} \\ \omega_{wz} \end{pmatrix} \quad (8)$$

Fig. 6 shows the the icon layer of the contact model with two multibody frame connectors from the Modelica Standard Library. The frames have to be connected to the center of the wheel and to the reference point on the rail which correspond to the points W and R in Fig. 4. The kinematical information of these two frames are required to evaluate (7) and (8).

The contact model does not have a state but represents a loop-closing element, which corresponds to the property that the wheel surface is constrained to be in touch with the rail surface. The constraint is formulated using the function from Sec. 2.1. As a result, the contact force in normal direction, i.e. normal to the plane through the point C according to Fig. 5 implicitly follows from the closed loop condition.

### 2.3 Hertzian Normal Contact

The pressure distribution  $p$  in the contact plane according to the Hertzian theory [21, (3.65)] is given by

$$p(x, y) = \frac{3f_n}{2\pi ab} \sqrt{1 - \frac{x^2}{a^2} - \frac{y^2}{b^2}}, \quad (9)$$

where  $f_n$  represents the absolute value of the normal force, while  $a$  and  $b$  denote the semi-axes of the contact ellipse and follow from

$$a = m \sqrt[3]{\frac{3(1 - \kappa^2)f_n}{E(A + B)}}, \quad b = n \sqrt[3]{\frac{3(1 - \kappa^2)f_n}{E(A + B)}}. \quad (10)$$

Beside Young's modulus  $E$  and Poisson's ratio  $\kappa$  that are assumed to be identical for wheel and rail, (10) uses auxiliary terms that are determined by

the curvature of the wheel  $\rho_{wx}$  and the rail  $\rho_{rx}$  measured in the plane normal to the  $x$ -axis and the wheel curvature in the plane normal to the  $y$ -axis  $\rho_{wy}$ :

$$A = \rho_{wx} + \rho_{rx}, \quad B = \rho_{wy}, \quad \vartheta = \arccos \frac{A - B}{A + B}.$$

$m$  and  $n$  are coefficients depending on elliptical integrals and specify the eccentricity of the contact ellipse. Tab. 2 is an extraction of [21, Tab. 3.4] to give an impression on their quantitative values.

$\vartheta$	$0^\circ$	$0.5^\circ$	$1^\circ$	$10^\circ$	$45^\circ$	$90^\circ$
m	$\infty$	61.4	36.89	6.604	1.926	1
n	0	0.1018	0.1314	0.3112	0.604	1

Table 2: Hertzian parameters for the contact ellipse.

In [5], it is proposed to introduce a weighed mean value of the curvatures in (10) that corresponds to (6), e.g. to consider the wheel curvature

$$\bar{\rho}_{wy} := \frac{\int_{\bar{s}} \rho_{wy}(s) \exp\left(\frac{\Delta}{\alpha}\right) ds}{\int_{\bar{s}} \exp\left(\frac{\Delta}{\alpha}\right) ds}. \quad (11)$$

Eq. (11) is implemented in the function presented in Sec. 2.1 as indicated by the output values given there.

### 2.4 Linear Tangential Contact

For vanishing slip *Kalker's*, i.e. for small values of  $\boldsymbol{\nu}$ , linear theory [8, Sec. 2.2.2] is valid so that the creep forces  $f_x$  and  $f_y$  in and the torque  $l_z$  normal to the  $xy$ -plane depend linearly on the slip  $\boldsymbol{\nu}$  under consideration of the shear modulus  $G$ :

$$\mathbf{f} = \begin{pmatrix} f_x \\ f_y \\ l_z \end{pmatrix} = \mathbf{C}\boldsymbol{\nu} \quad \text{with} \quad (12)$$

$$\mathbf{C} = -abG \begin{pmatrix} C_{11} & 0 & 0 \\ 0 & C_{22} & \sqrt{ab}C_{23} \\ 0 & -\sqrt{ab}C_{23} & abC_{33} \end{pmatrix}.$$

For the sake of demonstration Tab. 3 shows exemplary values of the coefficients that appear in (12).

### 2.5 Polach's Tangential Contact

According to *Polach* [14] the torque  $l_z$  in (12) is usually very small and can be neglected, while the

$b/a$	1	0.8	0.6	0.4	0.2	0.1
$C_{11}$	4.12	4.36	4.78	5.57	7.78	11.7
$C_{22}$	3.67	3.99	4.5	5.48	8.14	12.8
$C_{23}$	1.47	1.75	2.23	3.24	6.63	14.6
$C_{33}$	1.19	1.04	0.892	0.747	0.601	0.526

Table 3: Coefficients of *Kalker's* linear theory of rolling contact for  $\kappa = 0.25$  and  $a > b$  [8, E3].

influence of the spin  $\phi_z$  on the lateral creep force  $f_y$  may be of considerable importance. This is why his creep force law considers two lateral components:  $f_y^\circ$  denotes the creep force that originates from lateral slip and  $f_y^*$  is associated to the spin:

$$\mathbf{f} = \begin{pmatrix} f_x \\ f_y^\circ + f_y^* \\ 0 \end{pmatrix} \quad (13)$$

The tangential force in *Polach's* model is defined in the direction of the slip resultant  $\bar{\nu}$  taking the influence of the spin into account:

$$\bar{\nu} = \sqrt{\nu_x^2 + \bar{\nu}_y^2}, \quad (14)$$

$$\bar{\nu}_y = \begin{cases} \nu_y + a\phi_z & \forall |\nu_y + a\phi_z| > |\nu_y| \\ \nu_y & \forall |\nu_y + a\phi_z| \leq |\nu_y| \end{cases}.$$

In order to evaluate the resulting friction force  $\bar{f}$  it is postulated that the tangential stresses grow linearly with the distance from the leading edge until saturation is reached. Hence, the analytical integration of the assumed stress field over the contact patch leads to

$$\bar{f} = -\frac{2f_n\mu}{\pi} \left( \frac{k_a\varepsilon}{1 + (k_a\varepsilon)^2} + \arctan(k_s\varepsilon) \right), \quad (15)$$

where  $\mu$  denotes the friction coefficient,  $k_a$  and  $k_s$ ,  $k_s \leq k_a \leq 1$ , are reduction factors associated to the adhesion or the slip area, respectively. They have been introduced by *Polach* [15] in order to account for wet or polluted conditions.  $\varepsilon$  represents the gradient of the tangential stress in the area of adhesion at the leading edge of the contact patch and is related to the coefficients of *Kalker's* linear theory:

$$\varepsilon = \frac{G\pi ab C_{jj}}{4f_n\mu}, \quad C_{jj} = \sqrt{\frac{C_{11}^2\nu_x^2}{\nu_x^2 + \nu_y^2} + \frac{C_{22}^2\nu_y^2}{\nu_x^2 + \nu_y^2}}.$$

The subdivision of the  $\bar{f}$  into its two components corresponds to the slip partitions:

$$f_x = \bar{f} \frac{\nu_x}{\bar{\nu}}, \quad f_y^\circ = \bar{f} \frac{\nu_y}{\bar{\nu}}. \quad (16)$$

The remaining creep force component  $f_y^*$  is evaluated separately yielding

$$f_y^* = -\frac{\phi_z}{\bar{\nu}} \frac{9af_n\mu(1 + 6.3(1 - e^{-\frac{a}{b}}))}{16} \cdot \left[ \varepsilon^* \left( \frac{\delta^3}{-3} + \frac{\delta^2}{2} - \frac{1}{6} \right) + \frac{\sqrt{(1 - \delta^2)^3}}{3} \right], \quad (17)$$

where the tangential stress gradient due to spin  $\varepsilon^*$  and the abbreviation  $\delta$  are used:

$$\varepsilon^* = \frac{8Gb\sqrt{ab} k_a C_{23}}{3f_n\mu(1 + 6.3(1 - e^{-\frac{a}{b}}))} |\bar{\nu}_y|, \quad \delta = \frac{(\varepsilon^*)^2 - 1}{(\varepsilon^*)^2 + 1}.$$

For traction vehicles running on adhesion limit, the dependence of the friction coefficient on the slip velocity may be considered relating the maximum friction coefficient  $\mu_0$  to the limit friction coefficient  $\mu_\infty$  at infinite slip velocity introducing the parameter  $B$ :

$$\mu = \mu_0 \left[ \left( 1 - \frac{\mu_\infty}{\mu_0} \right) e^{-Bv_{sx}} + \frac{\mu_\infty}{\mu_0} \right]. \quad (18)$$

The above used constants  $k_a$ ,  $k_s$ ,  $\mu_0$ ,  $\mu_\infty$  and  $B$  are heuristic quantities that have been introduced to account for deviations from theory observed by measurements. Tab. 4 quotes *Polach* to give typical values.

conditions	$k_a$	$k_s$	$\mu_0$	$\mu_\infty$	$B$ [s/m]
dry	1.00	0.40	0.55	0.22	0.60
wet	0.30	0.1	0.30	0.12	0.20

Table 4: Typical model parameters for dry and wet conditions of the real wheel-rail contact [15].

## 3 Application

### 3.1 Project Background

Fig. 7 shows an experimental running gear in scale 1:5 that operates on the DLR roller rig in Oberpfaffenhofen. Unlike usual wheel-set configurations this running gear has independently rotating wheels each driven by one wheel hub motor. Two opposite front wheels are mounted together on a cranked beam, the two rear wheels mounted on their carrier constitute the identical second wheel pair unit. Each wheel pair unit is connected to the central frame having one rotational degree of freedom around the vertical axis.

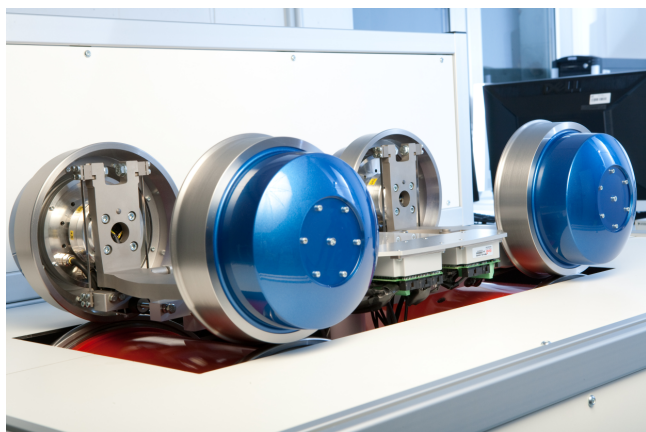


Figure 7: M 1:5 roller rig of DLR with experimental running gear for mechatronic guidance research.

That way the wheel pair units may perform independent yaw motions relative to the central frame. The central frame has two degrees of freedom with respect to the roller rig basis so it may move laterally and yaw.

The running gear has been designed in order to develop a new mechatronic guidance concept that allows for active steering. Significant wear, noise and weight reduction together with benefits that result from the low-floor configuration are goals that are on the agenda of the DLR internal project NGT [17].

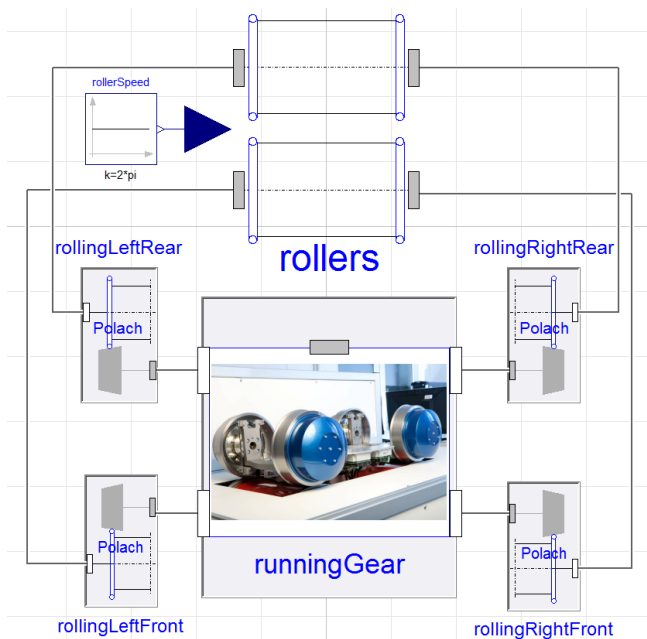


Figure 8: Diagram layer of the running gear model on the roller rig.

### 3.2 Model Particularities

In parallel to the experimental environment a simulation model of the running gear on the test rig has been established in Modelica. The diagram layer of the Modelica model is visualized in Fig. 8 while an animation of the running gear operating on the roller rig is presented in Fig. 9. Both environments together are supposed to support the development of advanced control design concepts.

However the aim of this paper is rather the modeling and the validation of the wheel-rail contact that has been introduced in Sec. 2 and is instantiated four times in Fig. 8. The contact model named *Polach* uses two multibody frame connectors to be linked to the center of the wheel and to the center of the associated roller. Since the contact model represents a loop-closing element, the contact force in normal direction, implicitly follows from the closed loop condition. The tangential forces are evaluated as described in Sec. 2.5.

Fig. 10 shows the main part of the menu used to parametrize the contact model. The wheel profile here is not a standard one, but can here be represented by a true conical shape. The control of the running gear is defined in such a way that only a repactive region of the wheel tread is in contact. Therefore the parameter  $r\_wheel\_FrontView$  is set to  $1 \cdot 10^{12}$  or infinity, respectively in Fig. 10.

The discussion in Sec. 2.1 is implicitly restricted to the contact of wheels to prismatic rails, which is different here due the geometry of the rollers with

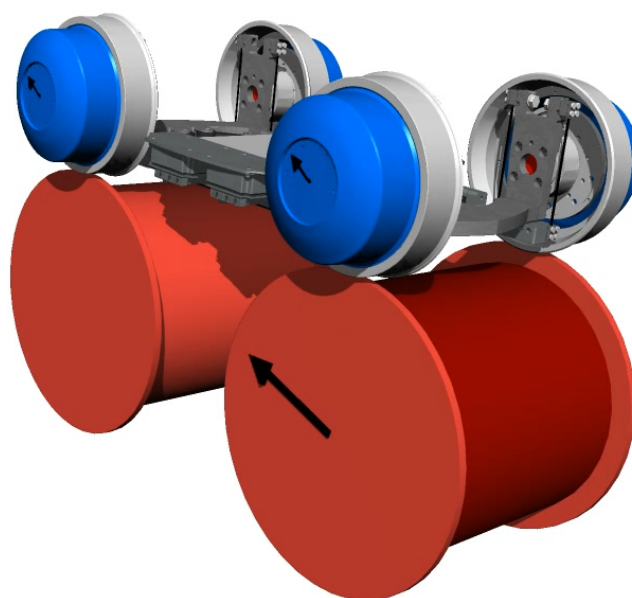


Figure 9: Animation of the running gear model.



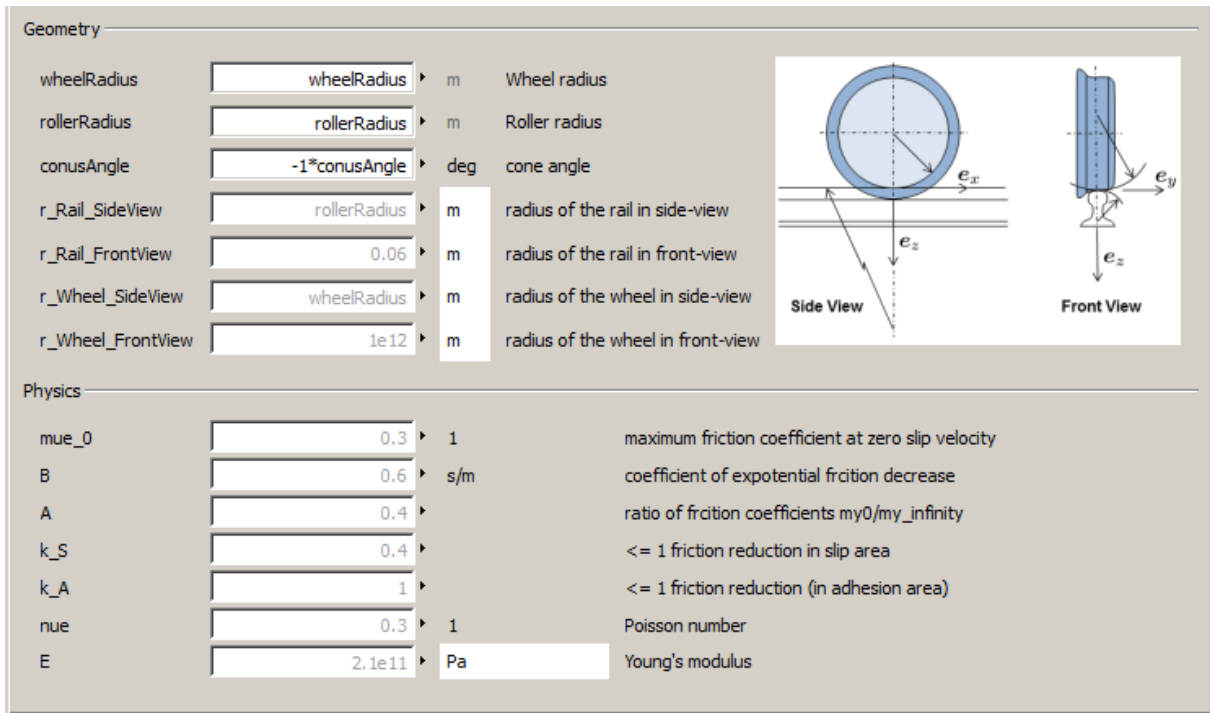


Figure 10: Parameter menu of the wheel-rail contact for conical wheel and curved roller profiles.

0.18 m radius. As soon as a wheel unit performs a yawing motion the two wheels leave the apexes of the associated rollers and slightly run downhill. This behavior is considered in the model but has been disregarded in Sec. 2, see [19, Sec. 2.2.2].

### 3.3 Results

A feature of the running gear are the force-torque sensors, that are assembled at the bearing of each wheel. Therefore the capability is given to compare simulation and measurement results with particular respect to the wheel-rail forces.

Fig. 11 to Fig. 13 show wheel-rail forces at the rear wheel on the right hand side. After the measurements have been low-pass filtered using a cut-off frequency of 20 Hz, the rotation frequencies of the roller and wheels still show up clearly, so that two narrow frequency bands of 0.2 Hz around these frequencies have been filtered out additionally. It is a current work field to eliminate or at least reduce the influence of the related disturbance sources.

The control of the running gear is set up in such a way that the running gear performs an artificial so-called hunting motion with 0.5 Hz frequency and a lateral amplitude of 8 mm. Whenever a conventional wheel-set is laterally excited e.g. by rail irregularities its dynamical response is a lateral

oscillation calling hunting. As long as the running velocity does not exceed a certain level, this motion is asymptotically stable. This is a desired dynamical property and a specific aspect of the wheel-rail profile design mentioned in Sec. 1.1. In addition hunting promotes wear not to be locally concentrated but distributed over a larger region of the wheel surface.

However independently rotating wheels do not show this passive behavior so that it has to be in-

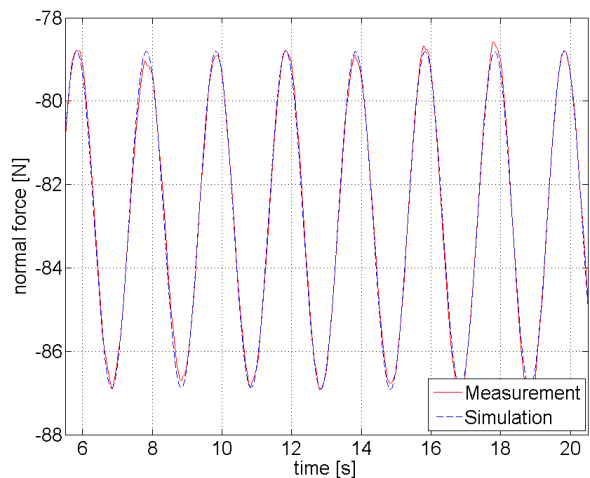


Figure 11: Comparison of normal wheel-rail force,  $v_0 = 6$  m/s, 0.5 Hz hunting frequency.

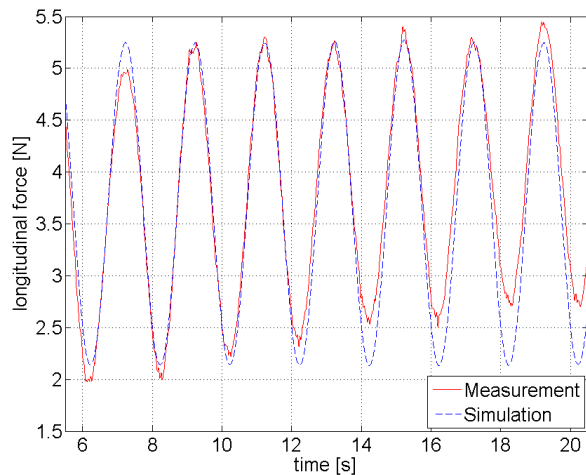


Figure 12: Comparison of longitudinal wheel-rail force,  $v_0 = 6$  m/s, 0.5 Hz hunting frequency.

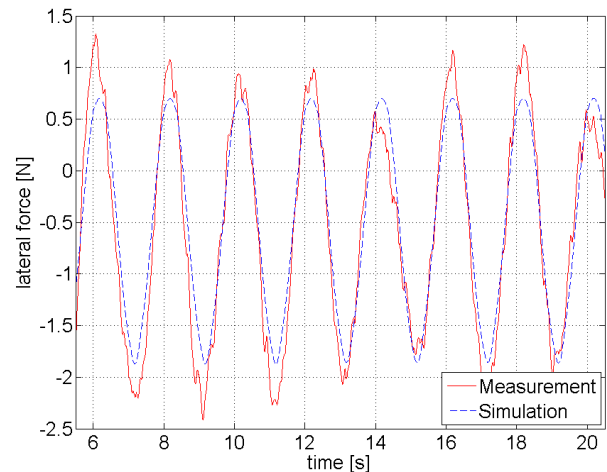


Figure 13: Comparison of lateral wheel-rail force,  $v_0 = 6$  m/s, 0.5 Hz hunting frequency.

troduced by feed-back control. The artificial hunting by control is the first proof a new mechatronic running gear concept has to stand, i.e. the mechatronic running gear has to perform at least as good as the conventional wheel-set design, before additional benefits could be approached.

Due to the hunting the normal wheel-rail force in Fig. 11 oscillates between 79 and 87 N. Measurements and simulation results correspond very good. The longitudinal forces in Fig. 12 show a long-wave deviation but are nevertheless rather close together. The values of the lateral forces in Fig. 13 are very small, which is actually intended by this specific running gear design. Therefore, the measurement tolerance of 0.25 N has to be considered when these results are assessed.

So far we are not able to measure the slip with sufficient accuracy. Therefore, the validation of the dependency of the forces on the slip is not possible today but will be tackled soon.

## 4 Conclusions and Outlook

From an intense literature review it has been concluded that a rigid elliptical single point contact with tangential force law according to *Polach* is expected to provide a well balanced compromise between accuracy and computational effort in order to establish a Modelica model of the wheel-rail contact.

Therefore the related theory has been summarized in Sec. 2. One additional refinement namely the quasi-elastic instead of the pure rigid contact

model has been introduced to guarantee a sufficient differentiability of the constraint equation.

This new Modelica wheel-rail contact is then applied to simulate the behavior of an experimental running gear on the scaled roller rig of DLR. The measured forces show a good agreement to the simulation results. A necessary advancement concerns the availability of slip measurements. Full scale applications with standard wheel-rail profiles will be modeled in the near future.

The new DLR RailwayDynamics Library, to which this wheel-rail contact model contributes a first cornerstone, is mainly intended to support the advanced observer and control design development and to facilitate multidisciplinary simulation tasks such as the interaction of running dynamics and drive train.

## Acknowledgements

This work was supported by BMBF (BMBF Förderkennzeichen: 01IS12022G), the German Federal Ministry of the Education and Research, within the ITEA 2 project Modrio.

## References

- [1] K. Knothe and S. Stichel. *Schienenfahrzeugdynamik*. Springer, Berlin, 2003.
- [2] Ingo Kaiser. Refining the modelling of vehicle-track interaction. *Vehicle System Dynamics*, 50(sup1):229–243, 2012.

- [3] O. Polach. Characteristic parameters of non-linear wheel/rail contact geometry. *Vehicle System Dynamics*, 48(S1):19–36, 2010.
- [4] H. Netter, G. Schupp, W. Rulka, and K. Schroeder. New aspects of contact modelling and validation within multibody system simulation of railway vehicles. *Vehicle System Dynamics*, 29(S1):246–269, 1998.
- [5] M. Arnold and H. Netter. Wear profiles and the dynamical simulation of wheel-rail systems. *Progress in Industrial Mathematics at ECMI*, 96:77–84, 1997.
- [6] G. Schupp, C. Weidemann, and L. Mauer. Modelling the contact between wheel and rail within multibody system simulation. *Vehicle System Dynamics*, 41(5):349–364, 2004.
- [7] J. Piotrowski and H. Chollet. Wheel–rail contact models for vehicle system dynamics including multi-point contact. *Vehicle System Dynamics*, 43(6-7):455–483, 2005.
- [8] J. Kalker. *Three-dimensional elastic bodies in rolling contact*, volume 2. Springer, 1990.
- [9] J. Piotrowski and W. Kik. A simplified model of wheel/rail contact mechanics for non-hertzian problems and its application in rail vehicle dynamic simulations. *Vehicle System Dynamics*, 46(1-2):27–48, 2008.
- [10] S. Iwnicki. *Handbook of railway vehicle dynamics*. CRC Press, 2006.
- [11] K. Knothe, R. Wille, and B.W. Zastrau. Advanced contact mechanics: Road and rail. *Vehicle System Dynamics*, 35(4-5):361–407, 2001.
- [12] J. Kalker. A fast algorithm for the simplified theory of rolling contact. *Vehicle System Dynamics*, 11(1):1–13, 1982.
- [13] K. Knothe. History of wheel/rail contact mechanics: from Redtenbacher to Kalker. *Vehicle System Dynamics*, 46(1-2):9–26, 2008.
- [14] O. Polach. A fast wheel-rail forces calculation computer code. *Vehicle System Dynamics*, 33:728–739, 2000.
- [15] O. Polach. Creep forces in simulations of traction vehicles running on adhesion limit. *Wear*, 258(7):992–1000, 2005.
- [16] E.A.H. Vollebregt, S. Iwnicki, G. Xie, and P. Shackleton. Assessing the accuracy of different simplified frictional rolling contact algorithms. *Vehicle System Dynamics*, 50(1):1–17, 2012.
- [17] J. Winter, E. Mittelbach, and J. Schykowski, editors. *RTR Special - Next Generation Train*. Eurailpress, DVV Media Group, 2011.
- [18] M. Arnold and H. Netter. Approximation of contact geometry in the dynamical simulation of wheel-rail. *Mathematical and Computer Modelling of Dynamical Systems*, 4(2):162–184, 1998.
- [19] H. Netter. *Rad–Schiene Systeme in differential-algebraischer Darstellung*. Number 352 in VDI–Fortschrittsberichte Reihe 12. VDI-Verlag, Düsseldorf, 1998.
- [20] W. Kortüm and P. Lugner. *Systemdynamik und Regelung von Fahrzeugen*. Springer-Verlag, Berlin, 1993.
- [21] K. Popp and W. Schiehlen. *Ground Vehicle Dynamics*. Springer, 2010.



# Human-Nature Interaction in World Modeling with Modelica

Rodrigo Castro<sup>1,2</sup>, Peter Fritzson<sup>3</sup>, François Cellier<sup>4</sup>, Safa Motesharrei<sup>5</sup>, Jorge Rivas<sup>6</sup>

<sup>1</sup>Department of Environmental Systems Science  
ETH Zurich, Switzerland

<sup>2</sup>Department of Computer Science, University of Buenos Aires  
and CIFASIS-CONICET, Argentina

<sup>3</sup>Department of Computer and Information Science  
Linköping University, SE-581 83 Linköping, Sweden

<sup>4</sup>Computer Science Department  
ETH Zurich, Switzerland

<sup>5</sup>National Socio-Environmental Synthesis Center (SESYNC)  
Annapolis, MD, 21401, USA

<sup>6</sup>Department of Political Science, University of Minnesota  
Minneapolis, MN 55408, USA

[rodrigo.castro@usys.ethz.ch](mailto:rodrigo.castro@usys.ethz.ch), [peter.fritzson@liu.se](mailto:peter.fritzson@liu.se), [fcellier@inf.ethz.ch](mailto:fcellier@inf.ethz.ch),  
[ssm@umd.edu](mailto:ssm@umd.edu), [jorgerodrigorivas@gmail.com](mailto:jorgerodrigorivas@gmail.com)

## Abstract

It is our predicament that we live in a finite world, and yet we behave as if it were infinite. Steady exponential material growth with no limits on resource consumption and population is the dominant conceptual model used by today's decision makers. This is an approximation of reality that is no longer accurate and started to break down. The World3 model, originally developed in the 1970s, includes many rather detailed aspects of human society and its interaction with a resource-limited planet. However, World3 is a rather complex model. Therefore it is valuable for pedagogical reasons to show how similar behavior can be also realized with models that are much simpler. This paper presents a series of world models, starting with very simple exponential growth and predator-prey systems, then investigates a minimal human-nature model, Handy, and ends with a brief account of the World3 model. For the first time, a simple human-nature interaction model is made available in Modelica that distinguishes between dynamics of Elite and Commoner social groups. It is shown that Handy can reproduce rather complex behavior with a very simple model structure, as compared to that of world models like World3.

*Keywords: Modelica, Simulation, Population Dynamics, World Models, Human-Nature Models.*

## 1 Introduction

Dynamic modeling can be applied to the human population and its interaction with the earth system. Certain types of such models are called World models.

Perhaps the earliest study indicating that human population would eventually reach limits to growth is that presented by Richard Malthus in 1798 in his famous work *An Essay on the Principle of Population* [4]. A catastrophe in the form of population collapse was considered as a possible consequence of uncontrolled growth that would hit the limits of food production causing massive famine.

A series of simulation models were developed mainly during the 1970's, with some later updates, aiming at understanding the complexity of the interactions between global societies and their physical environment, searching for the conditions that would lead to a collapse and possible measures that can avoid it. They formed the category of so-called World Models, typically of considerable generality and complexity, spanning several subsystems (demography, energy, economy, industry, agriculture, etc.) at varied levels of detail.

## 2 Growth without Limitations

In this first example, we shall show the dynamics of human populations in an idealized ecological system without limitations. We assume that a population contains  $P$  individuals. In general, the rate of change of the number of individuals is the difference between the population birth rate and its death rate:

$$\dot{P} = \text{birthrate} - \text{deathrate} \quad (1)$$

where  $\dot{P}$  means  $dP/dt$ , the rate of change of the population stock. It is natural to assume that the birth and death rates are proportional to the size of the population:

$$\begin{aligned} \text{birthrate} &= g \cdot P \\ \text{deathrate} &= d \cdot P \end{aligned} \quad (2)$$

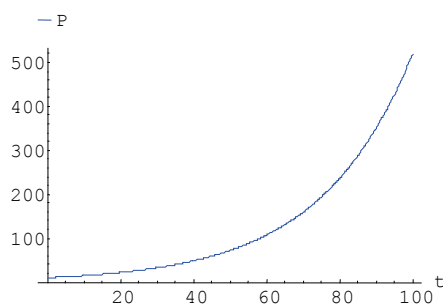
where  $g$  is the growth (birth) factor and  $d$  the decrease (death) factor for the population. These factors can be assumed to be constant if the same amount of food per individual is available independent of the size of the population, and if no sudden lethal events (disease, war) affect the population massively. These assumptions are of course valid only for a reasonably limited size of the population, since food supply is never infinite in a closed system and epidemics, pandemics, and wars sometimes break out. Putting (1) and (2) together gives:

$$\dot{P} = (g - d) \cdot P \quad (3)$$

Solutions to this equation yield an exponentially increasing population if  $(g - d) > 0$  or an exponentially decreasing one if  $(g - d) < 0$ . We represent this equation in the model class `PopulationGrowth` below:

```
class PopulationGrowth
  parameter Real g = 0.04 "Growth factor";
  parameter Real d = 0.0005 "Death factor";
  Real P(start=10) "Pop. size, init 10";
equation
  der(P) = (g-d)*P;
end PopulationGrowth;
```

The model is simulated and the population size  $P$  is plotted below:



**Figure 1.** Exponential growth of a population with unlimited food supply.

As expected, Figure 1 shows an exponentially increasing population. We should, however, remember that exponential growth can never persist indefinitely in a closed system with a limited food supply.

## 3 A Predator-Prey Model with Limitations in Prey Animal Population

Now we will study a more interesting system consisting of interacting populations of predators and prey animals.

The simplest predator-prey model, the so-called Lotka-Volterra model [3], consists of two population stocks. We may think of predators as foxes and preys as rabbits.

The rabbit population of size  $R$  is assumed to have an unlimited food supply. Therefore equation (3) applies regarding the birth rate of the rabbit population, with a positive growth term  $g_r \cdot R$ , where  $g_r$  is the growth factor for rabbits.

On the other hand, the fox population of size  $F$  feeds on the rabbits. The rabbit death rate can be assumed to be proportional to the number of foxes due to increased hunting pressure, and to the number of rabbits due to the higher probability of success in hunting, giving a rabbit death rate due to foxes of  $d_{rf} \cdot F \cdot R$ , where  $d_{rf}$  is the death factor of rabbits due to foxes. Putting the birth and death rates together gives the total rabbit population change rate  $\dot{R}$  as defined by equation (4) below:

$$\dot{R} = g_r \cdot R - d_{rf} \cdot F \cdot R \quad (4)$$

The growth of the fox population of size  $F$  is proportional to the rate of rabbits consumed, i.e., those that die because they were hunted by foxes, which is the death rate term  $d_{rf} \cdot F \cdot R$  mentioned in the previous equation. The efficiency of growing foxes from rabbits is determined by the growth factor  $g_{fr}$ , giving a fox population growth rate term  $g_{fr} \cdot d_{rf} \cdot R \cdot F$ . The fox population also has an intrinsic death rate  $d_f \cdot F$  proportional to the size of the fox population by the fox death factor  $d_f$ . By combining these two terms, we obtain equation (5):

$$\dot{F} = g_{fr} \cdot d_{rf} \cdot R \cdot F - d_f \cdot F \quad (5)$$

where  $\dot{F}$  is the total change rate of the fox population.

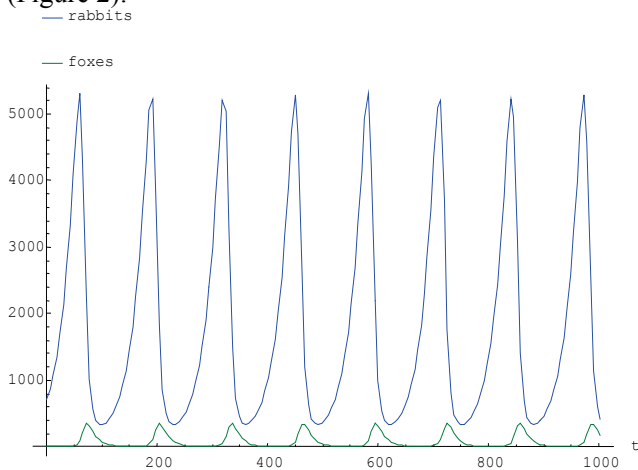
We have created a Modelica [2][8] class below called `LotkaVolterra` that includes both equations (4) and (5) along with all the variables and constant proportionality factors. The two state variables  $R$  and  $F$  for the sizes of rabbit and fox populations are represented by the Modelica variables `rabbits` and `foxes`:

```

class LotkaVolterra
  parameter Real g_r =0.04    "Natural
  growth rate for rabbits";
  parameter Real d_rf=0.0005  "Death rate
  of rabbits due to foxes";
  parameter Real d_f =0.09    "Natural
  deathrate for foxes";
  parameter Real g_fr=0.1     "Efficency
  in growing foxes from rabbits";
  Real rabbits(start=700) "Rabbits, (R)";
  Real foxes(start=10)    "Foxes, (F)";
equation
  der(rabbits) = g_r*rabbits -
    d_rf*rabbits*foxes;
  der(foxes)   = g_fr*d_rf*rabbits*foxes -
    d_f*foxes;
end LotkaVolterra;

```

The model is simulated, and the sizes of the rabbit and fox populations as a function of time are shown below (Figure 2):



**Figure 2.** Number rabbits—prey animals, and foxes—predators, as a function of time simulated from the PredatorPrey model.

The Lotka-Volterra model has a rather special property: the solution variables, i.e., the rabbit and fox population sizes, do not approach constant steady-state levels (unless they start at those equilibrium levels, which is rarely the case). Instead, they exhibit oscillatory equilibrium. The shape of these oscillations, very characteristic for Lotka-Volterra models, corresponds rather well to several experimental population studies of predators and prey animals. However, this does not prove these partly inductive model equations to be correct, only that they seem to have certain approximate physical significance.

## 4 A Minimal Human-Nature Dynamical Model with Economic Stratification

Simplified approaches to World models can exhibit interesting behavioral patterns while keeping model complexity low enough to enable intuitive comprehension. One such simplified model is Handy (Human and Nature Dynamical model) [9] developed jointly by researchers at the University of Maryland and the University of Minnesota<sup>1</sup>.

The LotkaVolterra model (Section 3) was the original inspiration behind Handy, which is not a world model in the typical sense, although, in principle, it can be applied to a homogenous world.

Handy has four differential equations describing the evolution of its state variables: Commoner population (*commoners*), Elite population (*elite*), regenerating natural resources (*nature*), and accumulated wealth (*wealth*). Human population plays a role analogous to that of predators, and nature plays the role of the resource preyed upon.

An interesting feature of Handy is that it introduces the accumulation of economic *wealth*, and divides the human population into rich and poor according to their unequal access to available wealth.

This new variable explains why human societies can undergo an irreversible collapse, while animal populations show cyclic changes (or reversible collapses).

Social inequality is not only explicitly considered but also plays a key role in the sustainability analyses of the model. This makes Handy the first model of its kind that studies the impacts of inequality on the fate of societies, a capability seldom found even in complex world models.

Handy establishes a useful general framework that allows carrying out “thought experiments” about societal collapse scenarios and the changes that might avoid them.

The model is a very strong simplification of the human-nature system, which results in many limitations. Despite its simplicity, such a model is easy to understand and offers a more intuitive grasp of underlying dynamical phenomena compared to more complex and less aggregated models.

We briefly introduce a much more advanced class of world model called `World3` in Section 5 of this paper. This model (the compiled version) features 41 state variables and 245 algebraic variables. It captures many

<sup>1</sup> The Handy model has been publicly available since 2010 in the form of unpublished reports and presentations, deposited on the web site of its authors:

[http://www.atmos.umd.edu/~ekalnay/#publications\\_based](http://www.atmos.umd.edu/~ekalnay/#publications_based).

aspects of the human system and ecology, and it has been calibrated against empirical data.

#### 4.1 Handy Model Equations, Description, and Modeled Societies

The Handy model divides the total population into two classes: Commoners and Elites, of population sizes `commoners` and `elite`, respectively. Population grows through birth rates `birthRComm` and `birthRElite`, which are constants. The population shrinks through death rates `deathRComm` and `deathRElite`, which are modeled as functions of `wealth`.

The state variables of the model are described in Table 1, additional variables in Table 2, and the parameters of the model in Table 3.

A single stock (variable), `nature` represents an amalgamation of nonrenewable and renewable resources, including “regenerating” resources such as forests, soils, and animal and fish stocks, etc.

The dynamics of the model are defined by the following four differential equations:

$$\begin{aligned} \text{commoners}' &= \text{birthRComm} \cdot \text{commoners} - \text{deathRComm} \cdot \text{commoners} \\ \text{elite}' &= \text{birthRElite} \cdot \text{elite} - \text{deathRElite} \cdot \text{elite} \\ \text{nature}' &= \text{natureRegen} \cdot \text{nature} \cdot (\text{natureCap} - \text{nature}) \\ &\quad - \text{depletFactor} \cdot \text{commoners} \cdot \text{nature} \\ \text{wealth}' &= \text{depletFactor} \cdot \text{commoners} \cdot \text{nature} \\ &\quad - \text{consRComm} - \text{consRElite} \end{aligned}$$

which in Modelica are expressed as:

```
der(commoners) = birthRComm*commoners -
deathRComm*commoners;
der(elite) = birthRElite*elite -
deathRElite*elite;
der(nature) =
natureRegen*nature*(natureCap-nature) -
depletFactor*commoners*nature;
der(wealth) =
depletFactor*commoners*nature - consRComm-
consRElite;
```

The equation for `nature` includes a regeneration term `natureRegen*nature*(natureCap-nature)` and a depletion term `depletFactor*commoners*nature` intended to also include degradation in nature caused by pollution.

The regeneration term is synthetic (i.e., not directly physical) following an s-shaped form<sup>2</sup> parameterized to resemble physically realistic results.

It exhibits exponential growth for low values of `nature` (because `natureCap-nature` is almost constant for small `nature`), reaches its maximum at `nature=natureCap/2`, and becomes small when `na-`

`nature` approaches `natureCap` (maximum size of `nature` in absence of depletion).

The depletion term includes a rate of depletion (pollution) per worker `depletFactor` making it proportional to both `nature` and `commoners` (labor). The economic activity of Elite is modeled to represent supervisory functions with no direct influence on the extraction of resources or the production of wealth. The underlying concept is that “only commoners produce.” Handy does not model the impacts of technological change, considering that technology through history has proven to produce both increases and decreases in resource use efficiency. Thus, a simplification is made assuming that these effects cancel each other out.

Accumulated Wealth (`wealth`) increases with production (`depletFactor*commoners*nature`) and decreases according to the consumption rates of the Elites and the Commoners, `consRComm` and `consRElite`, respectively.

**Table 1.** Main state variables of the Handy model.

Name	Description	Unit	Typical initial value(s)
commoners	Commoner population size	Number of people	100
elite	Elite population size	Number of people	0, 1, 25
nature	Nature's stock	EcoDollars	<code>natureCap</code> = 100
wealth	Accumulated wealth	EcoDollars	0

The additional variables of the model are described in Table 2. Two of these variables describe consumption. The consumption of the Commoners, `consRComm`, is given by a subsistence salary per capita `subsSal` times the working population. Elites earn a salary that is `ineqFactor` times larger than that of Commoners, intended to simulate class inequality, and is considered time-independent for any given scenario. This is reflected in the following equations:

$$\begin{aligned} \text{consRComm} &= \min(1, \text{wealth} / \text{wealthMin}) \cdot \text{subsSal} \cdot \text{commoners} \\ \text{consRElite} &= \min(1, \text{wealth} / \text{wealthMin}) \cdot \text{subsSal} \cdot \text{elite} \cdot \text{ineqFactor} \end{aligned}$$

which in Modelica are expressed as:

```
consRComm = min(1, wealth/wealthMin) *
subsSal*commoners;
consRElite = min(1, wealth/wealthMin) *
subsSal*elite*ineqFactor;
```

Both consumption rates, `consRComm` and `consRElite`, are subject to a “famine” minimum threshold for accumulated wealth before famine, `wealthMin`. The consumption rates are linearly reduced down to zero by the `min()` terms in the above equations, when `wealth` falls below `wealthMin`.

The death rates of the Commoners and the Elite are defined by the following equations:

<sup>2</sup> This is the well-known “logistic equation” used in many domains of life sciences to represent exponential growth followed by smooth saturation.



$$\begin{aligned}
 deathRComm &= deathRnormal + \max(0, 1 - consRComm) \\
 &\quad / (subsSal \cdot commoners) \cdot (deathRfamine - deathRnormal) \\
 deathRElite &= deathRnormal + \max(0, 1 - consRElite) \\
 &\quad / (subsSal \cdot elite) \cdot (deathRfamine - deathRnormal)
 \end{aligned}$$

The carrying capacity (`carryingCap`) of population, i.e., the population that can be sustained by the re-growth of nature, and its corresponding maximum carrying capacity (`carryingCapMax`), attainable under certain conditions, are defined by these equations:

$$\begin{aligned}
 carryingCap &= natureRegen / depleteFactor \cdot \\
 &\quad (natureCap - subsSal \cdot eta / depleteFactor)
 \end{aligned}$$

$$carryingCapMax = natureRegen / (eta \cdot subsSal) \cdot (natureCap / 2)^2$$

The threshold for minimum wealth (`wealthMin`), the total population, and a dimensionless quotient `eta` used above are defined as follows:

$$\begin{aligned}
 wealthMin &= consWorkerMin \cdot commoners + \\
 &\quad ineqFactor \cdot consWorkerMin \cdot elite \\
 eta &= (deathRfamine - birthRComm) / \\
 &\quad (deathRfamine - deathRnormal)
 \end{aligned}$$

$$totalPopulation = elite + commoners$$

**Table 2.** Additional variables of the Handy model for Human and Nature Dynamics.

Name	Description	Unit
deathRComm	Commoners' death rate	Number of people/year
deathRElite	Elites' death rate	Number of people/year
consRComm	Consumption rate of commoners	EcoDollars/year
consRElite	Consumption rate of Elites	EcoDollars/year
eta	Dimensionless quotient	
carryingCap	Carrying capacity (of the whole system)	Number of people
carryingCapMax	Maximum carrying capacity	Number of people
totalPopulation	Total population	Number of people

The parameters of the model are described in Table 3.

**Table 3.** Parameters of the Handy model for Human and Nature Dynamics.

Name	Description	Typical value(s)
deathRnormal	Healthy normal (minimum) death rate	0.01
deathRfamine	Famine (maximum) death rate	0.07
birthRComm	Commoner birth rate	0.03, or 0.065
birthRElite	Elite birth rate	0.03, or 0.02
subsSal	Subsistence salary per person	0.0005
consWorkerMin	Minimum required consumption per worker	0.005
natureRegen	Nature's regeneration rate	0.01

natureCap	Nature's carrying capacity	100
ineqFactor	Inequality factor between elite and commoners	1, 10, or 100
depletFactor	Depletion (production) factor	6.67E-6, or 6.35E-6, or 13E-6

There are three dimensions for quantities in the model: Population (either Commoners or Elites), in units of people, Nature or Wealth, in units of EcoDollars, and Time, in units of years. All other parameters and functions in the model carry units that are compatible with these dimensions.

All of the above definitions appear in the HandyBase Modelica model below, which is a partial model that is inherited by the following actual simulated models.

```

partial model HandyBase "Equations,
variables, and parameters for Handy"
  // All parameters, default values for
egalitarian society
  parameter Real birthRComm = 0.03 "Birth
Rate of Commoners No people/year";
  parameter Real birthRElite = 0.03 "Birth
Rate of Elite. No of people/yr";
  parameter Real natureRegen = 0.01
"Nature's regeneration factor.";
  parameter Real natureCap = 100.0
"Nature's capacity. ecoDollars";
  parameter Real subsSal=0.0005
"Subsistence Salary/Worker.
Dollars/worker";
  parameter Real consWorkerMin = 0.005
"Minimum required Consumption per
Worker. Dollars/worker";
  parameter Real depletFactorEq =
0.00000667
"Rate of depletion (pollution) per
worker at Equilibrium. 1/Worker*year";
  parameter Real deathRnormal = 0.01
"Healthy Death Rate. people/year";
  parameter Real deathRfamine = 0.07
"Famine Death Rate. people/year";
  parameter Real ineqFactor = 0
"Inequality in consumption level for
Workers and Non-Workers. Does not play a
role when elite=0";

  Real depletFactor "Rate of depletion per
worker. 1/Worker*year";
  Real commoners "Population of Commoners.
Number of people";
  Real elite "Population size of Elite.
Number of people";
  Real nature "Natural stock (renewable
and nonrenewable). ecoDollars";
  Real wealth "Accumulated wealth.
EcoDollars";
  Real wealthMin "Minimum threshold accum
wealth before famine. EcoDollars";
  Real deathRComm "Death Rate for
Commoners. Number of people/year";
  Real deathRElite "Death Rate for Elite.
Number of people/year";
  
```

```

Real consRComm "Consumption Rate of
Commoners. Dollars/year";
Real consRElite "Consumption Rate of
Elite. Dollars/year";
Real eta "Derived quotient expression.
Dimensionless";
Real carryingCap "Carrying Capacity.
Number of people";
Real carryingCapMax "Maximum Carrying
Capacity. Number of people";
Real totalPopulation "Total population:
elite and commoners. No people";

```

#### equation

```

der(commoners) = birthRComm*commoners -
deathRComm*commoners;
der(elite) = birthRElite*elite -
deathRElite*elite;
der(nature) = natureRegen*nature *
(natureCap-nature) -
depletFactor*commoners*nature;
der(wealth) = depletFactor* commoners *
nature - consRComm - consRElite;
deathRComm = deathRnormal + max(0,
1-consRComm / (subsSal*commoners)) *
(deathRfamine-deathRnormal);
deathRElite = deathRnormal + max(0,
1-consRElite / (subsSal*elite)) *
(deathRfamine-deathRnormal);

consRComm = min(1, wealth/wealthMin) *
subsSal*commoners;
consRElite = min(1, wealth/wealthMin) *
ineqFactor*subsSal*elite;
wealthMin = consWorkerMin*commoners +
ineqFactor*consWorkerMin*elite;

eta = (deathRfamine - birthRComm) /
(deathRfamine-deathRnormal);
carryingCap = natureRegen /depletFactor
*(natureCap - subsSal*eta/ depletFactor);
carryingCapMax = natureRegen /
(eta*subsSal) * (natureCap/2)^2;
totalPopulation = elite + commoners;

```

#### initial equation

```

nature = natureCap;
wealth = 0;
end HandyBase;

```

## 4.2 Types of Societies, Simulation Methodology and Scenarios

Handy has been applied to three types of societies:

- *Egalitarian society*—with no elites, i.e., the elite population `elite = 0`. Scenario models `HandyEgal1` to `ModelEgal4`.
- *Equitable society*—with workers (commoners) and elites (non-workers), where both groups earn the same per person, i.e., `ineqFactor=1`. Scenario models `HandyEquit1` to `HandyEquit5`.
- *Unequal society*—with commoners and elites, where elites consume more per capita than commoners, i.e., `ineqFactor > 1`. Scenario models `HandyUnEq1` to `HandyUnEq4`.

Several scenarios will be studied for each kind of society. The default values for parameters and initial values for state variables used in these scenarios are those presented in Table 1 and Table 3.

The thought experiments are performed as simulations, the results of which are presented in Sections 4.3 and 4.5, respectively. Within each type of society, different scenarios are studied by varying the rate of depletion per worker called `depletFactor` (short for depletion factor).

For each type of society, an equilibrium state can be reached, in which all derivatives go to zero, and consequently, all the system's variables settle into a steady state.

Under that condition, the carrying capacity can be maximized if nature's regeneration rate is maximal. According to the equation for nature, the latter is satisfied when `nature=natureCap/2`. The depletion factor `depletFactor` is at the optimal level `depletFactorEq`, when it produces a steady state with a maximum sustainable population.

The scenarios below explore the consequences of different types of societies as they behave optimally or sub-optimally, determined by the depletion factor.

## 4.3 Simulation Scenarios of an Egalitarian Society

For this type of society, the optimal value for the depletion factor that maximizes the carrying capacity (`carryingCap`) is `depletFactorEq=6.67E-6` (derived analytically). Below is the `HandyEgalitarianBase` model that is being inherited by all `Egal` models.

```

partial model HandyEgalitarianBase
"Egal - Scenarios with Egalitarian
Society (No-Elite)"
extends HandyBase;
initial equation
elite = 0; // For All "Egal" scenarios
commoners = 100.0;
end HandyEgalitarianBase;

```

*Scenario Model HandyEgal1: "Soft landing to Equilibrium"*

```

model HandyEgal1 "Scenario Egal1: Soft
Landing to Equilibrium"
extends HandyEgalitarianBase;
equation
depletFactor = depletFactorEq;
end HandyEgal1;

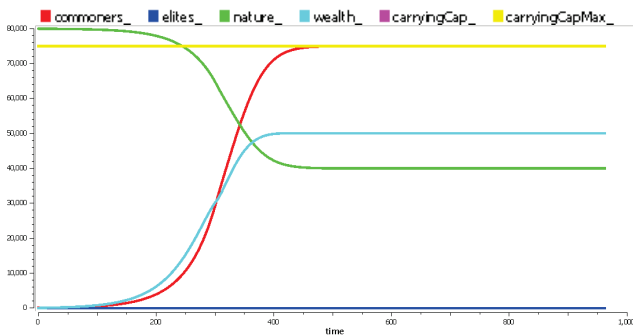
```

By making `depletFactor=depletFactorEq` the results in Figure 3 below are obtained:

```

plot({commoners_, elites_, nature_,
wealth_, carryingCap_, carryingCapMax_})

```



**Figure 3.** Soft landing to equilibrium, with optimal nature depletion factor. Simulation of `HandyEgal1`.

In this example, maximum nature regeneration can support a maximum sustainable depletion rate (pollution) and a steady maximum population (`commoners`) equal to the carrying capacity (`carryingCap`).

*Scenario Model HandyEgal3: “Cycles of Prosperity and Collapse”*

In this scenario we increase the depletion intensity by selecting `depletFactor = 4 * depletFactorEq`.

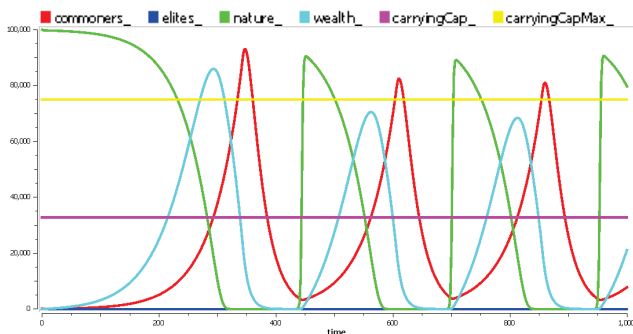
```

model HandyEgal3 "Scenario Egal3: Cycles
  of Prosperity and Collapse"
  extends HandyEgalitarianBase;
  equation
    depletFactor = 4 * depletFactorEq;
  end HandyEgal3;
    
```

The results obtained are shown in Figure 4 below:

```

plot({commoners_, elites_, nature_,
  wealth_, carryingCap_, carryingCapMax_})
    
```



**Figure 4.** Cycles of Prosperity and Collapse. Simulation of `HandyEgal3`.

The oscillatory mode exhibits “reversible collapses”. This scenario can be seen as belonging to a “limit cycle” for the values of the depletion factor. If `depletFactor` increases even further, the system changes into a different mode, one of “irreversible” collapse, with only one overshoot-and-collapse cycle. This is obtained in the scenario `HandyEgal4`.

*Additional Scenarios in an Egalitarian Type of Society*

There are also scenarios models `HandyEgal2` “Oscillatory Approach to Equilibrium” and `HandyEgal4` “Full

Collapse”, corresponding to `depletFactor = 2.5*depletFactorEq` and `depletFactor = 5.5*depletFactorEq`, respectively. The reader is encouraged to test these models.

#### 4.4 Simulation Scenarios of an Equitable Society—Workers and Non-Workers with Equal access to Wealth

In this society, Non-Workers and Workers have equal access to wealth, i.e., `ineqFactor = 1`. The non-workers’ privilege is that they get access to wealth without having to work (in that sense, they are regarded as the Elites). The initial population of non-workers is `elite = 25`. As before, we look for an equilibrium situation where the depletion factor is set to its optimal value, which for this society is `depletFactor = depletFactorEq = 8.33E-6`. This produces the first scenario `HandyEquit1` model shown further below, an adaptation of the `HandyEquitableBase` model which is inherited by all `Equit` scenario models.

```

partial model HandyEquitableBase
  "Equit Scenarios: Equitable society
  (with Workers and Non-Workers)"
  extends HandyBase(ineqFactor = 1,
    depletFactorEq = 0.00000833);
  initial equation
    commoners = 100.0;
  end HandyEquitableBase;
    
```

*Scenario HandyEquit1: “Soft Landing to Optimal Equilibrium”*

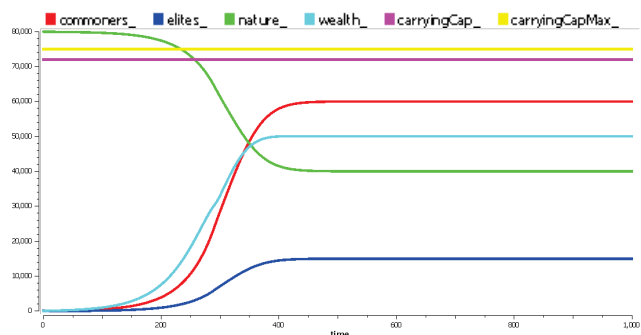
```

model HandyEquit1
  "Equitable society Scenario Equit1: Soft
  Landing to Optimal Equilibrium"
  extends HandyEquitableBase;
  initial equation
    elite = 25;
  equation
    depletFactor = depletFactorEq;
  end HandyEquit1;
    
```

We simulate and plot:

```

plot({commoners_, elites_, nature_,
  wealth_, carryingCap_, carryingCapMax_})
    
```



**Figure 5.** Equilibrium in the presence of workers (Commoners) and non-workers (Elites) with equal consumption per person. Simulation of `HandyEquit1`.

In Figure 5, the sum of worker and non-worker populations reaches the maximum carrying capacity. The optimal depletion factor in this society is bigger than that of an egalitarian society analyzed in the previous section. This is due to an increased amount of production required from the workers to sustain the non-workers.

*Scenario HandyEquit2: “Oscillatory Approach to Equilibrium”*

If the depletion factor is further increased to  $\text{depletFactor} = 2.64 * \text{depletFactorEq}$  we obtain an oscillatory behavior. The results are shown in Figure 6

```

model HandyEquit2
  "Equitable society Scenario Equit2:
  Oscillatory Approach to Equilibrium"
  extends HandyEquitableBase;
  initial equation
    elite = 25;
  equation
    depletFactor = 2.64 * depletFactorEq;
end HandyEquit2;

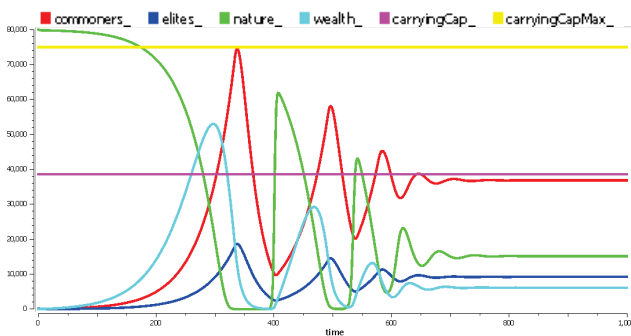
```

We simulate and plot:

```

plot({commoners_, elites_, nature_,
wealth_, carryingCap_, carryingCapMax_})

```



**Figure 6.** Oscillatory convergence to equilibrium with tolerable overshoot. Simulation of HandyEquit2.

Now, the carrying capacity (`carryingCap`) is lower than the maximum carrying capacity, but the total population approaches a steady state value after an oscillatory phase.

*Additional Scenarios in an Equitable Type of Society*

There are also scenarios HandyEquit3 “Cycles of Prosperity, Overshoot and Collapse”, HandyEquit4 “Full Collapse”, and HandyEquit5 “Preventing a Full Collapse by Decreasing Average Depletion per Capita”.

The reader is encouraged to also try these models, which include experimenting with different values of the depletion factor, and in the case of HandyEquit5, increasing the ratio of non-workers to workers.

#### 4.5 Simulation Scenarios of an Unequal Society—Elites and Commoners with Different Access to Wealth

The unequal society appears to be closer to the status of our current world. The inequality factor `ineqFactor` is made to vary from 10 to 100 in the unequal scenarios.

We will reproduce a pair of experiments intended to show the effects of changing birth rates and inequality as a means for moving from an unsustainable to a sustainable mode of behavior.

```

partial model HandyUnEquitableBase
  "Uneq Scenarios: Unequal Society (with
  Elite and Commoners)"
  extends HandyBase(ineqFactor = 100,
    depletFactorEq = 0.00000667,
    birthRComm = 0.03, birthRElite = 0.03);
end HandyUnEquitableBase;

```

*Scenario HandyUneq2: “Full Collapse”*

In this scenario the inequality factor is set very high: `ineqFactor = 100`. Also, the initial seed for the population of the Elite is set to `elite = 0.2` and a large depletion factor (including pollution effects) `depletFactor = 1E-4` is selected. The result is a full collapse with no recovery, as shown in Figure 7 below.

```

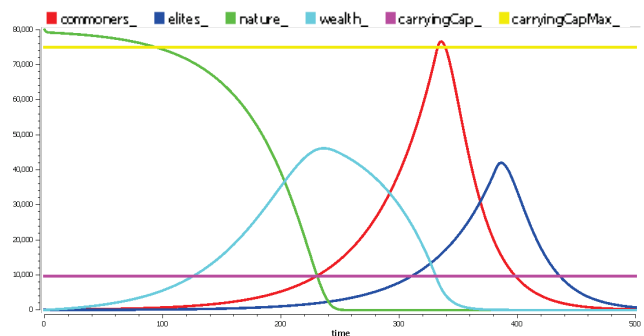
model HandyUneq2
  "UnEquitable society Scenario Uneq2:
  Type-II Collapse (Full Collapse)"
  extends HandyUnEquitableBase(ineqFactor
  = 100,
    depletFactorEq = 0.00000667,
    birthRComm = 0.03, birthRElite = 0.03);
  initial equation
    elite = 0.2;
    commoners = 100.0;
  equation
    depletFactor = 15 * depletFactorEq;
end HandyUneq2;

```

```

plot({commoners_, elites_, nature_,
wealth_, carryingCap_, carryingCapMax_})

```



**Figure 7.** Full collapse due to over-depletion and high levels of inequality (`ineqFactor=100`). Simulation of HandyUneq2.

As soon as the Commoners' population surpasses the carrying capacity, *wealth* starts to decline and never again recovers. Before disappearing, Elites remain insensitive to the wealth's fast decrease for a long period after the Commoners' population starts its massive decline. This is possible due to the unequal access to wealth, which is a hundred times larger than that of the Commoners.

*Scenario HandyUneq3: "Soft Landing to Optimal Equilibrium"*

In this scenario several parameters and initial values are changed with respect to the previous case.

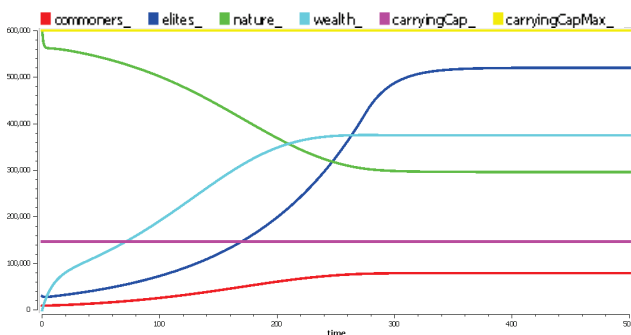
- Inequality is reduced by a factor of ten, yielding `ineqFactor = 10`.
- The depletion factor is set to its equilibrium value, derived analytically: `depletFactor = depletFactorEq = 6.35E-6`, much lower than in the previous scenario.
- The initial values for the population are set as `commoners=10 000` and `elite=3000`.
- The birth rates are assumed to be controllable and selected as `birthRComm=0.065` and `birthRElite=0.02`.

The results are shown below:

```

model HandyUnEq3
  "Unequitable society UnEquit3: Soft
  Landing to Optimal Equilibrium"
  extends HandyUnequitableBase(ineqFactor
  = 10, depletFactorEq = 0.00000635,
  birthRComm = 0.065, birthRElite =
  0.02);
  initial equation
    elite = 3000.0;
    commoners = 10000.0;
  equation
    depletFactor = depletFactorEq;
  end HandyUnEq3;
plot({commoners_, elites_, nature_,
wealth_, carryingCap_, carryingCapMax_})

```



**Figure 8.** Unequal society reaching a sustainable equilibrium with moderate inequality (`ineqFactor=10`) and birth rate control. Simulation of `HandyUnEq3`.

The new set of parameters produces a sustainable equilibrium in an unequal society. It represents an example of implementation of policies that simultaneously limit inequality and allow birth rates to remain below critical levels. In this model inequality and birth rate are separate but simultaneous measures. In fact, in the real world high birth rates in poor societies are often a consequence of inequality.

*Additional Scenarios for an Unequal Type of Society*

There are also scenarios `HandyEquit1` "Collapse with Recovery of Nature" and `HandyEquit4` "Oscillatory Approach to Equilibrium", which perform experiments with different values of the depletion factor, birth rates, and values of initial populations.

## 5 A Short Look at the World3 Model

`World3` is a rather complex world model and therefore, its realm, scope, and objectives are quite different from those of `Handy`. The compiled version contains 41 state variables and 245 algebraic variables, with the same number of equations, representing many facets of human society and global ecological and economic dynamics.

The model is available as part of the `SystemDynamics.WorldDynamics` library [1], developed by François Cellier and his students. They made a documented Modelica version of `World3` by translating the original model from its version in Stella.

The `SystemDynamics.WorldDynamics.World3` model is represented as a Modelica package that implements Meadows et al `World3` model. `World3` was first described in the 1972 book *Limits to Growth: A Report for the Club of Rome's Project on the Predicament of Mankind* [5]. The book has seen two later editions, one published in 1992 (20-year edition), and the most recent one published in 2004 (30-year edition) [7]. Each of these new editions were accompanied by an updated model. The model in this Modelica `World3` version is the newest model discussed in the 2004 edition *Limits to Growth: The 30-Year Update*.

Whereas Jay Forrester listed his entire mathematical model in his book *World Dynamics*, Meadows et al in *Limits to Growth* only discussed the results obtained using their model. The mathematical model itself was not listed. The main reason was to make the book more accessible to a wider public. Another reason is that `World3` is considerably more complex than `World2`, and consequently, a thorough discussion of all aspects of the model would have taken up much more space in the book. Instead, the authors published a separate book of 637 pages in 1974: *Dynamics of Growth in a Finite World* [6] that describes all facets of the model.

Since the World3 model is rather complex, it was subdivided into 13 different sectors (i.e., submodels) describing aspects of the following: 1) arable land dynamics, 2) food production, 3) human ecological footprint, 4) human fertility, 5) human welfare index, 6) industrial investment, 7) labor utilization, 8) land fertility, 9) life expectancy, 10) non-recoverable resource utilization, 11) pollution dynamics, 12) population dynamics, and 13) service-sector investment.

In the overall main model, one submodel of each class is placed on the screen, and the individual submodels are connected appropriately (Figure 9).

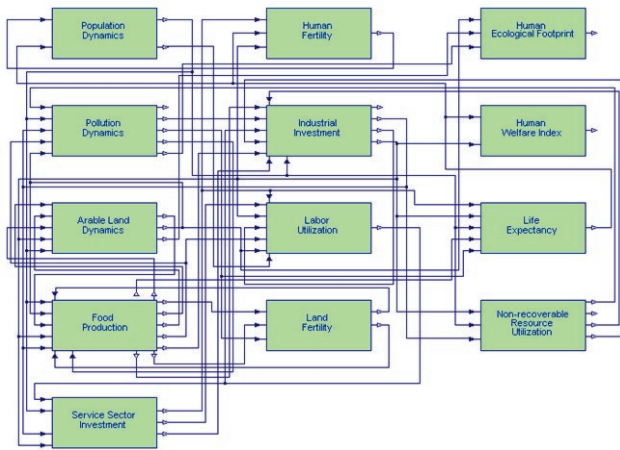


Figure 9. Overall World3 model.

The submodel `World3.Land_Fertility`, for example, is depicted below in Figure 10.

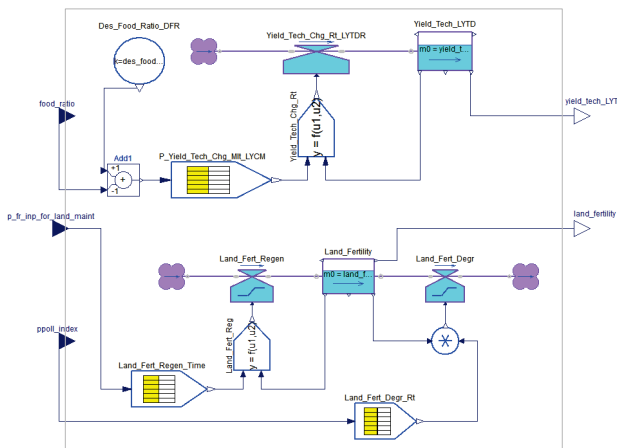


Figure 10. A submodel of World3: `SystemDynamics.World3.Land_Fertility`.

This submodel has 3 Inputs and 2 outputs. Two state variables, `Land_Fertility` and `Yield_Tech`, are represented by means of “Level” blocks, which have the ability to accumulate. Levels play the role of the state variables in a differential equation. Levels are in turn driven by the input and/or output “Rates” connected to the left and right sides, respectively, determining the velocities at which the accumulated quantity increases

or decreases. These Rates, in turn can be functions of other variables, including the state of the Level itself. Said functions are defined and interconnected following a block diagram approach, resorting to a vast library of pre implemented blocks such as Non-Linear Functions, Table Lookup Functions, Gains, Multipliers, etc.

The “Clouds” connected to the rates have only a pictorial purpose, expressing the fact that a source and/or a sink always exists with the physical ability to provide and/or absorb material without limits. The latter can be considered the conceptual “boundaries” of the model, as no detail is given for the processes taking place in material supply or consumption at these ends.

Whereas the World2 model lumps the entire population into a single state variable, the World3 model offers a demographic population dynamics model that distinguishes between children and adolescents, young adults of child-bearing age, older adults who are still integrated into the work force, and the retired population.

The capital investment is subdivided into investments in the military/industrial complex, in the service sector, and in agriculture.

Both the natural resources and pollution models have been upgraded by including changes in technology as factors influencing the depletion of resources and the release of pollutants. This is meaningful as improved technology may enable us to use the available resources more efficiently, and may also make it possible to produce goods in a cleaner fashion.

*Scenarios with World3*

The book *Limits to Growth: The 30-Year Update*, describes eleven scenarios based on different sets of assumptions. These scenarios are also part of the Modelica World3 model. We shall briefly discuss three of these scenarios:

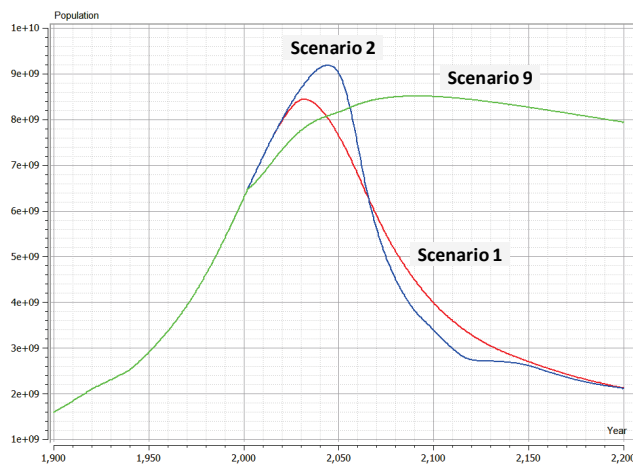
- Scenario1 is the so-called “Standard Run”. This is the original World3 model with basic assumptions and without any adjustments.
- Scenario2 is the same as Scenario1, but with twice as much non-recoverable resources initially available, i.e., more oil, coal, metal, etc.
- Scenario9 is based on Scenario2 but applies a number of measures to avoid collapse and to try to transform the human society into a sustainable one.

The three scenarios are simulated between year 1900 and 2200.

Unfortunately, both Scenario1 and Scenario2 lead to population collapse rather quickly, as depicted in Figure 11. Doubling the non-renewable resources in sce-

scenario2 does not help at all (it delays the collapse, but makes it steeper). The population increases a little further, but the subsequent collapse is even more severe, primarily due to increased levels of pollution. The current developments in our world unfortunately seem to be rather close to Scenario2.

If a collapse indeed should occur, will it be as quick as depicted in Figure 11? That must not necessarily be the case. The World3 model assumptions may not be completely valid during such a stressful transition period. Political decisions, rationing policies, etc., may lead to decreasing resource consumption and create a slower, more controlled contraction to a sustainable level of consumption.



**Figure 11.** The world population according to Scenarios 1, 2, and 9 in the World3 model.

How can a collapse be avoided? Scenario9 applies the following aggressive technology development plan:

- Increased industrial resource efficiency, i.e., a reduction in the use of non-renewable material and energy, a 4% resource decrease/year.
- Arable land protection, e.g., to decrease and prevent land erosion, and to preserve existing land.
- Agricultural yield enhancement, a 4% increase/year.
- Pollution reduction, a 4% decrease/year.

Additionally, a program for birth control is implemented with an average number of two children per family. The scenario furthermore assumes that there is a certain capital cost to implement these technologies, and that 20 years are required for their full implementation.

The Scenario9 simulation to year 2200 shows a sustainable society. However, if the simulation continues to the year 2500, we observe that it is hard to maintain sustainability with a large human population combined with high standards of living. If the size of the population is gradually decreased back to its level of about a hundred years ago, the chances for being able to maintain high standards of living are much better.

Another issue is model validity. A number of assumptions behind the World3 model may not be valid that far into the future. However, one fact is certain: the human race must, in the long run, live sustainably on planet Earth due to limitations in resources, energy, and space. Consumption overshoots like the current one can only be temporary.

## 6 Conclusion

The Handy model has the remarkable feature of providing a minimal structure helpful for intuitive understanding of the human-nature interaction. It can produce a rich variety of dynamical modes, leading to non-trivial scenarios. This combination is seldom found in most world models, where the quest for including more details comes at the price of lack of intuitiveness.

Handy is very useful as a “conceptual” model that can trigger varied interpretations and discussions comparing qualitatively different scenarios.

However, the goal of Handy is not to make quantitative short-term forecasts of the world state, but to assess qualitatively the impacts of various factors such as inequality and depletion on the long-term behavior of human societies.

The explicit stratification of social classes guides the interpretations to an essential playground needed to discuss the fate of human population.

Also worth mentioning is the fact that human societies developed a degree of cultural and technological complexity that prevents treating them in the same way as an animal species.

Accumulation of wealth (human’s ability to save throughout generations) is a major distinction which allows for irreversible population collapses, whereas with predation only, the predator-prey type of models can only produce cyclic oscillations (i.e., partial collapses).

Several senior modelers have remarked that it is not reasonable to expect that any world model will correctly predict how the world will develop after the onset of collapse, as unpredictable social disruptions and/or restructurings would result that invariably invalidate most of the model’s assumptions, parameters and even structures that might have described pre-collapse phases in a credible way. Such social reshaping will have its dynamics determined by interactions between different social classes, and in that regard, again, the Handy approach of disaggregating them in at least two strata represents a valuable contribution to future world modeling.

Moreover, we want to draw the reader’s attention to the aggregation made in Handy of natural renewable and non-renewable resources into a stock variable. This

paves the way for attaining scenarios that look sustainable in the very long run (i.e., when simulating “to infinity”). Based on the principles of thermodynamics it can be shown that in the long run the maximum carrying capacity of humans on earth will be determined *only* by the remaining renewable resources (once we have depleted all the non-renewables for good).

If we consider renewable and non-renewable resources independently of each other (along with the commoners, the elites, and their inequality factor), we may be able to use the resulting world model to investigate scenarios that offer possible answers to an additional highly relevant question. How should we best use the non-renewable resources to attain a society with minimum inequalities, for when the time comes that humanity must rely on renewable resources only? To investigate such a goal, the inequality factor should be altered from a constant parameter to a variable. Although models presented in this paper are from different classes (minimal Handy vs. more complex, realistic world model, World3), their conclusions are similar. In the long run, not so far into the future, humanity must change to living sustainably on planet Earth. This change can occur either as a planned gradual transition, preserving well-functioning societies, or as a more disruptive, unplanned transition resulting in a less pleasant society with a reduced ecological capacity.

## 7 Acknowledgments

This work has been supported by the Swedish Governmental Agency for Innovation Systems (Vinnova) within the ITEA2 MODRIO project, and by the Swedish Research Council (VR).

## References

- [1] Cellier, François. World3 in Modelica: Creating System Dynamics Models in the Modelica Framework. In *Proceedings of the 6th International Modelica Conference*, Bielefeld, Germany. 2008.
- [2] Fritzson, Peter. *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*, 940 pages, ISBN 0-471-471631, Wiley-IEEE Press. Feb. 2004.
- [3] Lotka, Alfred J. *Elements of Mathematical Biology*. Dover Publications, New York, 1956.
- [4] Malthus, T. *An Essay on the Principle of Population*. Printed for J. Johnson, in St. Paul’s Church-Yard, London, 1798. (Reprint. Amherst, NY: Prometheus Books, 1998).
- [5] Meadows D., Meadows D., Randers J., and Behrens III W.W. *Limits to Growth: A Report for the Club of Rome’s Project on the Predicament of Mankind*. 205p. Universe Books, New York, 1972.
- [6] Meadows D., Behrens III W.W., Meadows D., Naill R.F., Randers J., and Zahn E.K.O. *Dynamics of Growth in a Finite World*. 637p. Wright-Allen Press, 1974.
- [7] Meadows D., Randers J., and Meadows D. *Limits to Growth: The 30-Year Update*. 368p. Chelsea Green, 2004.
- [8] Modelica Association. Modelica—A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.2 rev 2. Available at <http://www.modelica.org>, August, 2013.
- [9] Motesharrei S., Rivas J., and Kalnay E. A Minimal Model for Human and Nature Interaction. Accepted for publication (expected 2014) *Journal Ecological Economics*.



# 1D/2D Cellular Automata Modeling with Modelica

Victorino Sanz<sup>†</sup> Alfonso Urquia<sup>†</sup> Alberto Leva<sup>‡</sup>

<sup>†</sup> Dpto. Informática y Automática, ETSI Informática, UNED

Juan del Rosal, 16, 28040, Madrid, Spain

{vsanz,aurquia}@dia.uned.es

<sup>‡</sup> Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano

Piazza Leonardo da Vinci 32, 20133 Milano, Italy

leva@elet.polimi.it

## Abstract

Cellular Automata (CA) can be used to describe dynamic phenomena dependent of the spatial coordinates. This approach exhibits two main advantages: CA models are conceptually simple and can be simulated very efficiently. A new Modelica library named *CellularAutomataLib* is presented. It facilitates describing one- and two-dimensional CA in Modelica, and interfacing these CA models with other Modelica models. Simulation performance and large model support have been highest priority in the design of the library. To achieve these goals, the CA internal description is programmed in C and it is consequently hidden to the modeling environment, which is released from the burden of causalizing and manipulating the millions of equations that typically compose CA models. The library architecture and use are discussed in this manuscript. Two examples illustrate the library use: heat diffusion on a chip and spread of an epidemic disease. *CellularAutomataLib* is freely available at <http://www.euclides.dia.uned.es>.

*Keywords:* Cellular Automata, Hybrid Models, Modelica

## 1 Introduction

Cellular Automata (CA) are discrete and dynamic models initially proposed by John Von Neumann for the study of self-reproducing automata [1]. These models are represented as a grid of identical volumes, named cells, that can be in any finite number of dimensions [2]. The state of each cell in the automata is discrete, and it is updated at discrete time steps during the simulation following a transition function or rule. This rule constitutes a function of the current state of the cell and the state of its neighbors, and

defines the state of the cell for the next time step [3]. The neighborhood of a cell is usually composed of a selection of its surrounding cells, but not necessarily. It can be defined in different ways, such as the Moore's neighborhood that includes all the surrounding cells; the von Neumann's neighborhood that includes the cells adjoining the four faces of one cell; or the extended von Neumann's that also includes each cell just beyond one of the four adjoining cells [4].

Formally, CA can be defined as a tuple [5]:

$$CA : \langle T, X, \Omega, S, \delta, Y, \lambda \rangle$$

where  $T$  is the time base (isomorphic with  $\mathbb{N}$ );  $X$  is the input set;  $\Omega$  is the set of all input segments  $\omega$  (an input segment may be restricted to a domain  $T$ ,  $\omega : T \rightarrow X$ );  $S$  is the state that is the same for all cells because the cellular space is homogeneous;  $\delta : \Omega \times S \rightarrow S$  is the global transition function used to update the state of each cell ( $\delta(\omega, s_i) \rightarrow \delta_l(N_i)$ ,  $\delta_l$  is the uniform local transition function,  $s_i$  is the state of the  $i$ -th cell of the grid and  $N_i$  is the set of states that correspond to the neighborhood of the  $i$ -th cell, usually defined as a set of offsets from  $i$ );  $Y$  is the output set; and  $\lambda : S \rightarrow Y$  is the output function used to observe the state of the automata.

The application of CA in the study of systems is broad and diverse, mainly due to the simplicity of describing these kind of models (i.e., by describing the state of the cell, the initial state of the space and the transition rule) and the computational efficiency of their simulation. They have been used to model systems in medicine [6], architecture [7], chemistry [8], economics [9], biology [10], among many others [11].

The feasibility for describing CA models using the Modelica language was demonstrated by Fritzson [12]. He described the Conway's Game of Life model

in Modelica by representing the cellular space using a matrix of integer numbers. The initial conditions are set using a vector that contains the coordinates of the initially active cells. At discrete times, generated using a *sample* operator, the state of the automata is updated by iterating the whole matrix using *two for* loops, and using the transition function to update each individual cell. The model uses the Moore's neighborhood. In this model, the description of the cellular space and the evaluations of the transition function in each cell are coupled, diffculting its reutilization to describe other automata.

Another approach to describe CA models in Modelica was performed by the authors [13]. The *CellularPDEVS* package, distributed with the DESLib library [14, 15], supports the description of CA using the Parallel DEVS formalism [16, 17]. DESLib is freely distributed under the Modelica License 2, and can be downloaded from the Modelica Association website. The cellular space is represented as coupled Parallel DEVS models, and each cell is described as an atomic Parallel DEVS model. *CellularPDEVS* allows the user to focus on describing the behavior of the cell and the characteristics of the cellular space. The state of each cell corresponds to the state of the atomic Parallel DEVS model and can be represented using an arbitrarily complex Modelica data structure. The transition rule corresponds to the internal transition function of each cell, which can contain any Modelica algorithm. This approach facilitates the description of the model by making the simulation algorithm of the automata transparent to the user.

*CellularPDEVS* also facilitates the combination of CA models with other Modelica models. Inputs and outputs to the cellular space can be described using the external transition and output functions, respectively. These functions (i.e., internal transition, external transition and output) are used in the DEVS formalism to define the behavior of models. However, the performance and the scalability of this library are not satisfactory. The reasons are twofold. First, the size of CA models is typically in the order of hundreds of thousands and millions of equations. The translation of models with so large number of equations (when even possible) is time consuming and huge executable files are generated (see also discussion in [18]). Second, long event chains are executed to update the CA state. The complete model is reevaluated after executing each event in the event chain, which in most cases is unnecessary, degrading the simulation performance significantly.

A new library, named *CellularAutomataLib*, for describing CA models is presented in this manuscript. The objective of this new library is to preserve the characteristics of *CellularPDEVS*, in terms of facility to describe the behavior of the model and the characteristics of the space, and provide a solution for its main drawbacks (i.e., simulation performance and scalability). *CellularAutomataLib* is not based in the Parallel DEVS formalism. The CA model (i.e., the state of the cell and the transition function) is described as a C data structure (i.e., a C *struct*) and a function. Its simulation algorithm is also directly implemented into several C functions that are called from Modelica by using the external function interface provided by the language [19]. CA models defined in this way are not manipulated by the Modelica tool (Dymola in our case), which produces an smaller simulation code and avoids the reevaluation of the whole model after the treatment of an event in the CA. The user can focus on describing the behavior of the model and not the simulation algorithm. Also, the simulation of CA models automatically displays a graphical animation that is generated using Gnuplot [20]. *CellularAutomataLib* provides interface models that facilitate the connection of CA with other Modelica models. These interface models use user-defined external functions to translate the state of the cell into a standard Modelica data type that can be used in other models, and vice-versa. The library has been developed using Dymola FD1 2013 on an Intel Core i5 2.3GHz machine with 16GB of RAM and running Linux 3.11 x86\_64.

The structure of the manuscript is as follows. The architecture of the library and its design principles are detailed in Section 2. The procedure to develop new CA models using the library is described in Section 3. The description of the interface models used to combined CA with other Modelica models is given in Section 4. Two case studies of 2D CA models are presented in Section 5. Finally, some conclusions and future work ideas are given in Section 6.

## 2 Architecture of the Library

The architecture of *CellularAutomataLib* is shown in Fig. 1. The library is composed of the following models and packages: License (description of the license); User Guide (library documentation); Path\_gnuplot (path to Gnuplot binary, used to generate the graphical animation); Input\_Region model (used as interface between CA models);

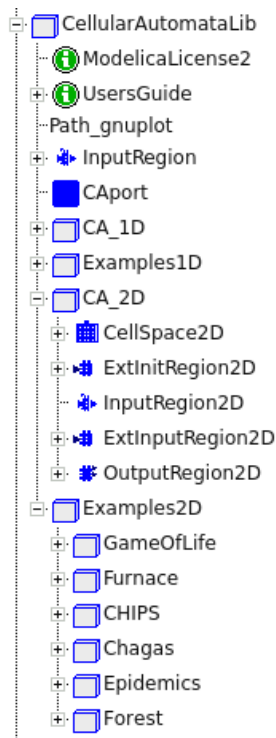


Figure 1: Architecture of *CellularAutomataLib*.

CAport connector (used to connect CA models with other models); CA\_1D package (models to describe 1D CA); Examples1D package (examples of 1D CA); CA\_2D package (models to describe 2D CA); Examples2D package (examples of 2D CA).

The library has been implemented using C functions that are called from Modelica functions using the external function interface. The basic functions that perform the simulation of the automata are included in the file `CellularAutomataLib.c`, which should not be modified by the user. These functions include the creation and initialization of the space, the simulation of a step or the reception of an external input, among others. The behavior of the model has to be described as external C code (i.e., `model.c`).

### 3 Development of New Models

A CA model in *CellularAutomataLib* is composed of one or several *cellular spaces*, that represent the 1D or 2D grid of cells, and some models, named *interface models*, used as interface between cellular spaces or between cellular spaces and other models. Cellular spaces and interface models include functions that call external C functions. These external C functions are used to describe the behavior of the

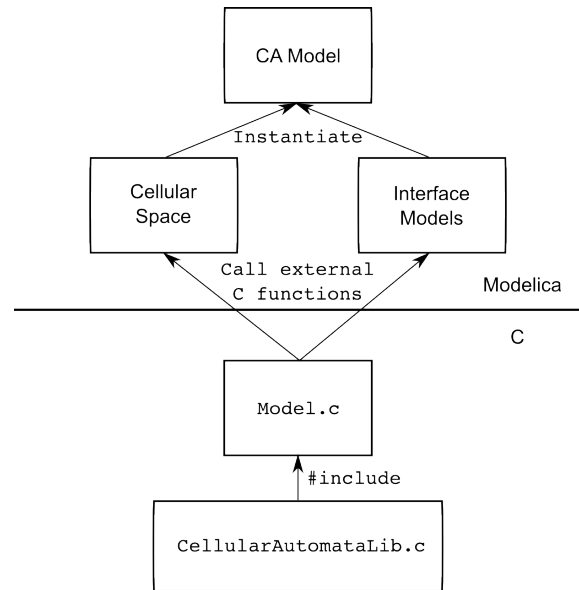


Figure 2: Relationship between Modelica and external C code in *CellularAutomataLib*.

models. The relationship between the external code and the Modelica code is summarized in Fig. 2. In this section the development of new cellular spaces is described. The use of the interface models is described in Section 4.

#### 3.1 Description of New Cellular Spaces

A cellular space in *CellularAutomataLib* is composed of the cellular space model and some functions. The cellular space model is a partial model that describes the one- or two-dimensional space represented by the automata. The parameters of the model are shown in Table 1. The cellular space model includes three replaceable functions: *Create*, that is used to create the cellular space, allocate memory for the cells and set them to the default state; *Initial*, that is used to initialize the cells indicated using the `init_cells` parameter and; *Rule*, that represents the transition function and is used to update the state of the cells at each simulation step.

The behavior of the cellular space is described by redeclaring these functions with functions that call external C functions (cf. Fig. 2). The description of these external functions is detailed below.

At the beginning of the simulation, the cellular space model creates the space using the *Create* function, and initializes the cells included in the `init_cells` parameter. After that, it performs periodic simulation steps every `Tstep`

Table 1: Parameters of cellular space models.

<i>Name</i>	<i>Description</i>
space_nrows	defines the number of rows of the space (only used in 2D spaces).
space_ncols	defines the number of columns of the space.
neighborhood	defines the topology of the neighborhood. It contains a list of the relative positions of the neighbors from the center cell.
n_inputs	defines the number of inputs received from interface models connected to the automata.
wrapped_borders	defines the boundary conditions. In 1D spaces it is either 1 for wrapped or 0 for non-wrapped. In 2D spaces it can be 0 for non-wrapped, 1 for wrapped only north-to-south, 2 for wrapped only east-to-west and 3 for wrapped in all directions.
Tstep	defines the interval between two steps in the simulation of the automata.
Initial_step	defines the time for performing the first simulation step.
plot_animation	defines if the graphical animation is generated (value 1) or not (value 0).
plot_range	defines the maximum value of the variable displayed in the animation. Thus, the displayed variable can be in the $[0, plot\_range]$ interval.
display_delay	defines a delay in the graphical animation that can be used to improve its visualization, which otherwise could be too fast to be observed.
init_cells	defines a list of coordinates of the cells that will be initialized at the beginning of the simulation.
name	defines a name for the automata that will be displayed in the graphical animation.

time, starting at  $time = Initial\_step$  (i.e.,  $sample(Initial\_step, Tstep)$ ).

### 3.2 Description of External C Functions

In order to facilitate the description of the behavior of new cellular spaces, the library includes a template file (named `draft.c`) that can be used to describe the required external C functions.

Following the formal specification of the automata, the user has to define the state variables that represent the state of the cells ( $S$ ) and the model behavior (i.e., the transition function  $\delta$ ) by reimplementing the functions included in the `draft.c` file, into a new file (e.g., `model.c`). The time base  $T$  is set using the parameters `TStep` and `Initial_step` of the cellular space model. The rest of the elements of the tuple  $(X, \Omega, Y, \lambda)$  are defined using the interface models.

As an example, the development of the Rule 30 model described by Wolfram [4] is presented. The transition function for this model is shown in Fig. 3.

The `draft.c` file can be used as a template to describe the behavior of the model. It has been renamed as `wolfram.c` for this example. The state of each cell is defined as an `int` value by modifying the `State` data type in the template. The default value for the cell state will be set using the `DefaultState` function, and so it has to be modified to set the

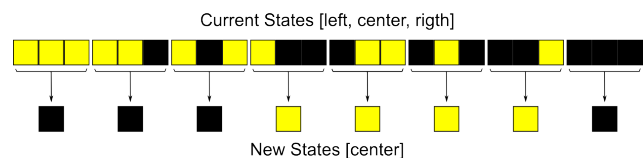


Figure 3: State transitions for the Rule 30 model.

default state to 0. The state of initialized cells will be set using the `InitialState` function, and so it has to be modified to set the initial state to 1. The transition function shown in Fig. 3 has to be implemented by modifying the transition function in the template. In order to automatically generate the graphical animation, the `Display` function in the template has to be modified to convert the state of the cell (i.e., the `State` data type) into a double value.

The cellular space for the Rule30 model is described in Modelica by extending the `CellSpace1D` model from `CellularAutomataLib`. The `Create`, `Initial` and `Rule` functions are redeclared using functions that call the external C functions defined in `wolfram.c`. The parameters for the Rule30 model are: `Space_ncols = 20`, `neighborhood = {-1,1}`, `wrapped_borders = 1`, `Tstep = 1`, `Initial_step = 0`, `plot_animation = 1`, `plot_history = 1`, `init_cells = 10`, `name = "Rule 30"`. The graphical

animation will be automatically generated using Gnuplot if `plot_animation` is set to 1. In this model, state 0 is displayed in black, and state 1 is displayed in yellow. The first 10 steps of simulation for the Rule 30 model are shown in Fig. 4 (the number of step is represented in the vertical axis).

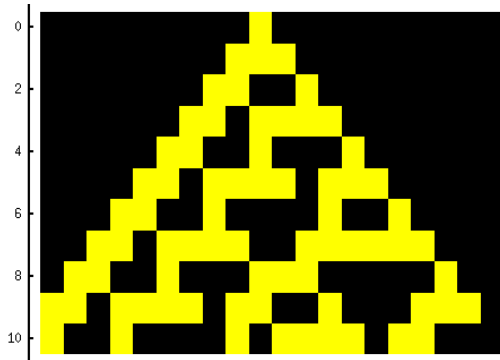


Figure 4: Simulation of the first 10 steps for the Rule 30 model.

## 4 Interfacing with Other Models

*CellularAutomataLib* includes several interface models that facilitate the combination of CA with other Modelica models. The inputs of the CA model ( $X, \Omega$ ) are described using *input* and *external input* region models. The outputs of the CA model ( $Y, \lambda$ ) are described using the *output* region model. Their behavior and use are detailed below.

### 4.1 Input Region

Cellular spaces can be combined to increase the modeling functionality of the library. This communication can be performed using the *Input\_Region* model. The same model can be used between 1D and 2D spaces.

The combination is performed by translating the state of some cells from one space as inputs for the other. The prototype of the transition function in C includes a vector of the received inputs, in order to allow the user to manage them. Each *Input\_Region* has associated an input identifier, set using the parameter `input_id`, that can be used as index for the vector of inputs of the transition function.

The *Input\_Region* model has two interface ports: *FROM* and *TO*. These interface ports are used to connect to the involved cellular spaces. The state of the cell  $[i, j]$   $i \in [RstartFrom:RendFrom]$ ,  $j \in [CstartFrom:CendFrom]$ , in the *FROM* space, is translated using

the `SetInput` function into an input for the cell  $[m, n]$   $m \in [RstartTo, RstartTo + (RendFrom - RstartFrom)]$ ,  $n \in [CstartTo, CstartTo + (CendFrom - CstartFrom)]$  in the *TO* space. `RstartFrom`, `RendFrom`, `CstartFrom`, `CendFrom`, `RstartTo` and `CstartTo` are parameters of the model. In 1D spaces, only the column parameters are used (i.e., `CstartFrom`, `CendFrom` and `CstartTo`). An additional parameter, named `column_1D_2D`, allows to use a 1D region as a column, instead of a row, of inputs for a 2D space. The communication is started at `time = comm_start` and is performed every `comm_rate` time.

The function `void SetInput(int Fspace, int Frow, int Fcol, int Tspace, int Trow, int Tcol, int input_id)` from the `draft.c` file can be used to redeclare the `SetInput` of this model.

### 4.2 External Input Region

Similarly to the *Input\_Region* model, the model *ExtInputRegion* can be used to set an input to a region of cells in the automata. In this case, the input is generated using an external signal instead of the state of the cells of other automata.

The model receives an external Real input signal through port *u*, which is used as input for a region of cells in the automata connected to port *TO*. As in the previous interface model, a 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). In 1D regions only the parameters referring to columns are considered. The input is assigned to the position `input_id` of the vector of inputs which is available for the user in the transition function. The external signal can be observed in the following ways (defined by the parameter `Input_type`), in order to be converted into an input:

- *Quantizer*: the input is set every time the value of the signal changes in a defined value or quantum.
- *Cross\_UP*: the input is set every time the value of the signal crosses a defined threshold in the upwards direction.
- *Cross\_DOWN*: the input is set every time the value of the signal crosses a defined threshold in the downwards direction.
- *Cross\_ANY*: the input is set every time the value of the signal crosses a defined threshold in any direction.

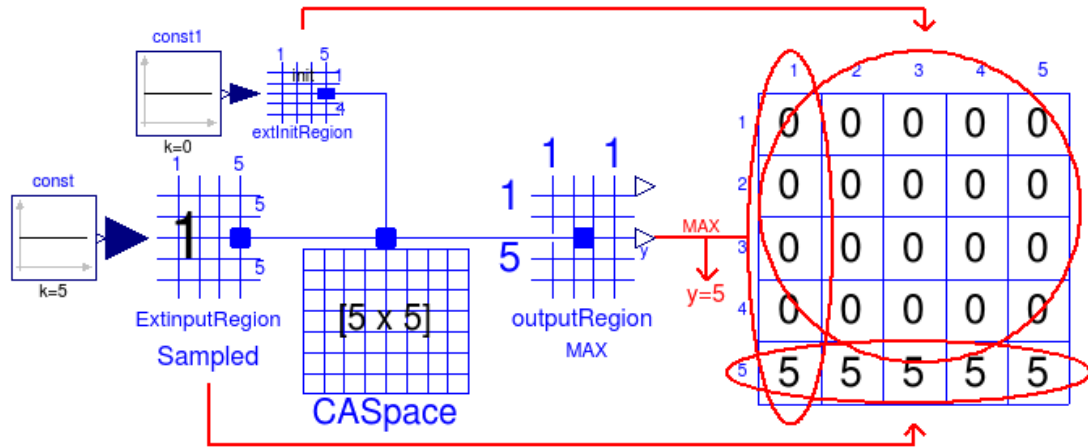


Figure 5: Example of behavior of external input region, external init region and output region models.

- *Sample*: the input is set periodically using the sample operator.

The signal is translated into an input using the function `ExtInput`, that can be redeclared using the void `ExtInput(int space,int row, int col, double value, int input_id)` function included in `draft.c`.

An example of external input is shown in Fig. 5. The `ExtInputRegion` model, connected to the CA `CASpace`, is used to set the state of the cells in the fifth row of the cellular space to the value of its input `const` (i.e., 5). The values of the states of the cells are graphically represented at the right of the figure.

### 4.3 External Init Region

This model can be used to set the initial state of a region of cells in the space using the value of an external signal. The model has an input port, named `u`, where a Real signal is received, and a port named `T0` that connects to the CA. This signal is translated, using the `ExtInit` function, into a cell state that will be used to initialize the cells in the region of the automata connected to port `T0`.

The 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). Only the column parameters are considered for 1D regions. The `ExtInit` function can be redeclared using the void `ExtInit(int space,int row, int col, double value)` function included in `draft.c`.

An example of external init is shown in Fig. 5. The `extInitRegion` model is used to initialize the state of the

cells in the rows 1 to 4 (and columns 1 to 5) with the value of its input `const1` (e.g., 0).

### 4.4 Output Region

The `OutputRegion` model can be used to observe the state of the cells in a region of the automata connected to port `FROM`. The state is translated into an output Real signal that can be used by other Modelica models. As in the previous interface models, a 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). In 1D regions only the parameters referring to columns are considered.

The model contains two output Real ports, `y` and `yM[Rend-Rstart+1,Cend-Cstart+1]` (being `yM[Cend-Cstart+1]` for the 1D case). Depending on the value of the parameter `Output_type`, the state is observed in different ways:

1. (AVERAGE): the value of `y` is calculated as the average value of the states of the cells in the `[Rstart : Rend,Cstart : Cend]` interval, or `[Cstart : Cend]` for 1D.
2. (MAX): the value of `y` is calculated as the maximum value of the states of the cells in the `[Rstart : Rend,Cstart : Cend]` interval, or `[Cstart : Cend]` for 1D.
3. (MIN): the value of `y` is calculated as the minimum value of the states of the cells in the `[Rstart : Rend,Cstart : Cend]` interval, or `[Cstart : Cend]` for 1D.

4. (MATRIX): the value of the state of the  $i, j$ -th cell in the space is assigned to  $yM[m, n]$  (where  $m = 1 : (Rend - Rstart + 1)$  and  $n = 1 : (Cend - Cstart + 1)$ ). In 1D, the  $i$ -th cell is assigned to  $yM[n]$  (where  $n = 1 : (Cend - Cstart + 1)$ )

The value of the state is translated into a Real value using the `ExtOutput` function, that can be redeclared using the double `Output(int space, int row, int col)` function included in `draft.c`.

An example of output region is shown in Fig. 5. The `outputRegion` model is used to calculate the maximum value (i.e., `Output_type = MAX`) among the states of the cells in the first column of the space. In the case shown in the figure, the output port `y` of `outputRegion` is set to 5.

## 5 Case Studies

`CellularAutomataLib` includes several examples of 1D and 2D models, whose purpose is to demonstrate the functionality of the library and to facilitate the development of new models. The modeler can use these examples as a base for constructing new CA. Two of these examples are described in this section: a model of heat transfer on a chip and a SIR (Susceptible Infected Removed) epidemic spread model.

### 5.1 Heat Transfer on a Chip

The model describes the flow of the heat generated in a chip by the execution of software instructions. Two heat transfer mechanisms are considered: heat diffusion in the chip surface and convective heat flow from the chip surface to the air. The model contains two bi-dimensional CA: one describes the chip and the other describes the air. The software execution is modeled using power sources located at certain points of the chip surface. These points correspond to the position of the circuit components (ALU, memory, etc.) that dissipate more heat.

The structure of the CA model is shown in Fig. 6. This model combines two cellular spaces, one for the chip (named `Chip`) and another for the air (named `Air`), with other Modelica models used to represent the sources of power (named `T+3S+N`, `Pg1` and `Pg2`). Two external input region models (named `Pext1` and `Pext2`) are used to combine the external sources of power with the `Chip` cellular space. Two input regions are used to represent the transfer of heat between chip and air (named `Chip2Air`), and vice-versa (named `Air2Chip`). An external init region (named `InitTemp`) is

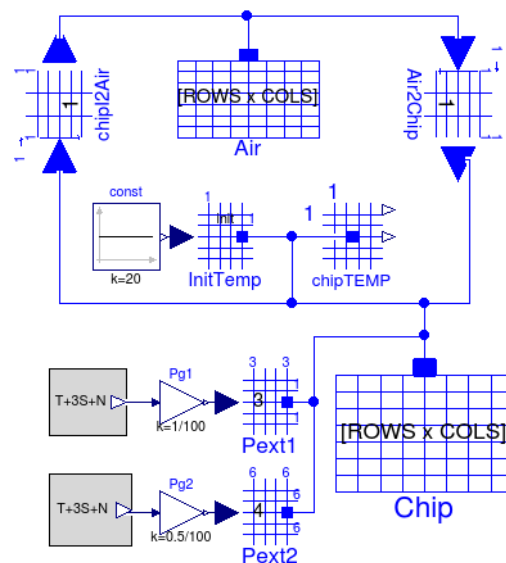


Figure 6: CA model of heat transfer on a chip.

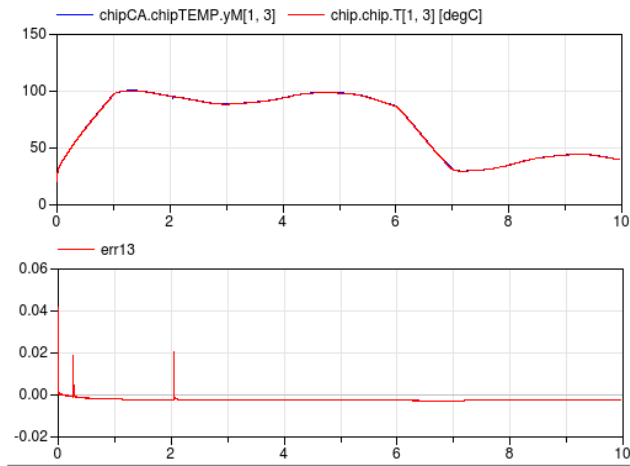
Table 2: Parameters of the chip model.

Name	Value	Unit	Description
gamma	100	$\frac{W}{m^2 \cdot K}$	Heat transfer coefficient
cp	710	$\frac{kg \cdot K}{m^3}$	Specific heat capacity
ro	2330	-	Average density
rows	10	-	Number of rows
cols	10	-	Number of columns
length	0.005	<i>m</i>	Layer length
width	0.005	<i>m</i>	Layer width
thickness	0.0001	<i>m</i>	Layer thickness
k	149	$\frac{W}{m \cdot K}$	Thermal conductivity

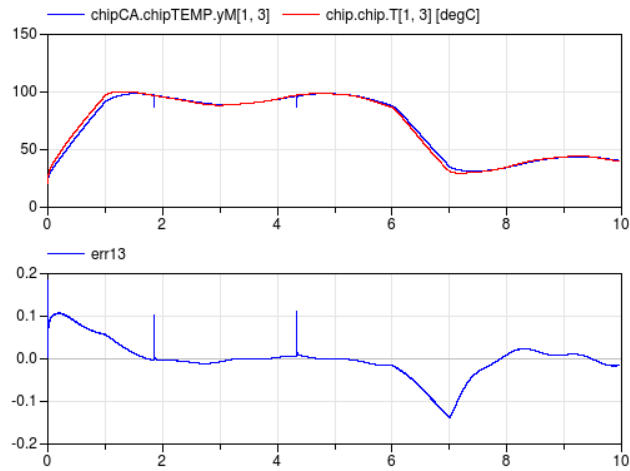
used to initialize the cells of `Chip` at 20°C, given by a constant source from the Modelica Standard Library. Finally, an output region model (named `chipTEMP`) is used to observe the evolution of temperatures in the chip.

The model has been implemented into a C file, named `chip.c`. The equations that describe the heat transfer have been implemented in the transition function of `Chip`. The `Air2Chip` model sets the temperature of the Air as an input for `Chip`, which is used to calculate the convection of heat from the chip to the air. The `Pext1` and `Pext2` set two inputs for `Chip` that are used as input heat flows in the equations.

The transition function calculates the evolution of temperatures in the chip using two alternatives: a forward Euler and a leap-frog integration algorithms. These are explicit integration algorithms that are easily



(a) Euler integrator



(b) Leap-frog integrator

Figure 7: Simulation results for cell [1,3] and error between Modelica and CA models: a) using Euler integration; and b) using Leap-frog integration.

included in the transition function of the CA. At each step of the simulation the transition function calculates an integration step and updates the values of the temperatures. The interval between steps using the forward Euler has to be 0.0001s in order to ensure stability. That interval can be increased to 0.001s using the leap-frog algorithm.

An analogous model has been developed using Modelica. In this model, the space has been discretized using finite volumes. In order to perform a comparison between the Modelica and the CA models, each volume will be represented by a cell in the CA.

Both models, Modelica and CA, have been simulated for 10s using the parameters shown in Table 2. The simulation results at the cell [1,3] and the error between the Modelica and CA approaches

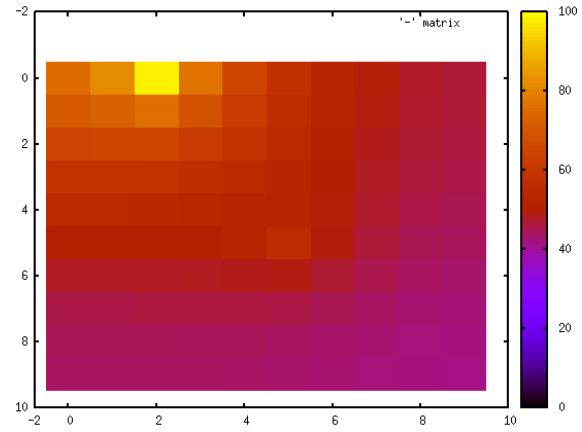


Figure 8: Capture of the graphical animation for the chip CA model.

Table 3: CPU time (in seconds) for integration of models during 10s of simulated time.

Model	Grid Side Size			
	10	50	100	200
Modelica (DASSL)	0.02	15.3	506	error
Modelica (EULER)	1.26	1225	5046	-
CA (EULER)	22	1670	6720	-
CA (LEAP-FROG)	2	173	711	2790

Table 4: Number of equations for the Chip model.

Model	Grid Side Size			
	10	50	100	200
Modelica	$1.2e^3$	$2.7e^4$	$1.1e^5$	$4.4e^5$
CA	$1.7e^3$	$2.5e^3$	$1e^4$	$4e^4$
CA without chipTEMP	70	70	70	70

are shown in Fig. 7. A capture of the graphical animation is shown in Fig. 8. The evolution of the simulation time with respect to the size of the grid is shown in Table 3. The simulation using EULER integration for the 200x200 grid was not performed and thus it does not appear in the table. The number of equations in the Modelica and CA models are shown in Table 4. If the chipTEMP output region model is removed from the CA model the number of equations is 70, independently of the grid size. The Modelica model rapidly reaches the maximum number of equations that can be efficiently handled by Modelica/Dymola, while the number of equations in the CA model remains lower. Note that Dymola



fails, due to an unknown internal error, to compile the Modelica model in a grid of 200x200 cells.

## 5.2 Epidemic Spread

The dynamics of epidemic spread are modeled in this example. This model was proposed in [21]. It is a SIR model where susceptible (S), infected (I) and recovered (R) individuals are considered. The evolution of the number of these individuals is defined by Eqs. (1), (2) and (3).

$$I_{ij}^t = (1 - \varepsilon) \cdot I_{ij}^{t-1} + v \cdot S_{ij}^{t-1} \cdot I_{ij}^{t-1} + S_{ij}^{t-1} \cdot \sum_{(\alpha, \beta) \in V} \frac{N_{i+\alpha, j+\beta}}{N_{ij}} \cdot \mu_{\alpha\beta}^{i,j} \cdot I_{i+\alpha, j+\beta}^{t-1} \quad (1)$$

$$S_{ij}^t = S_{ij}^{t-1} - v \cdot S_{ij}^{t-1} \cdot I_{ij}^{t-1} - S_{ij}^{t-1} \cdot \sum_{(\alpha, \beta) \in V} \frac{N_{i+\alpha, j+\beta}}{N_{ij}} \cdot \mu_{\alpha\beta}^{i,j} \cdot I_{i+\alpha, j+\beta}^{t-1} \quad (2)$$

$$R_{ij}^t = R_{ij}^{t-1} + \varepsilon \cdot I_{ij}^{t-1} \quad (3)$$

where  $V$  is the neighborhood of the  $(i, j)$  cell, and  $\mu_{\alpha\beta}^{i,j} = c_{\alpha\beta}^{(i,j)} \cdot m_{\alpha\beta}^{(i,j)} \cdot v$ , where  $c_{\alpha\beta}^{(i,j)}$  and  $m_{\alpha\beta}^{(i,j)}$  are the connection factor and the movement factor between the  $(i, j)$  cell and its neighbor cell  $(i + \alpha, j + \beta)$ , and  $v \in [0, 1)$  is the virulence of the epidemic. The parameter  $\varepsilon$  defines the portion of infected individuals that recover from the disease at each step.

This model has been programmed in a C file, named `epidemics.c`. The size of the cellular space has been set to 50x50 in order to validate its results with the ones presented in [21]. Only the cell in the center of the space (i.e., position [25,25]) is initialized at the beginning. The parameters are set using the values:  $\varepsilon = 0.4$ ,  $v = 0.6$ ,  $c = 1$  and  $m = 0.5$ . The Moore's neighborhood is used. The CA model includes the cellular space model and three output region models (see Fig. 9), used to sum the values of the state variables (S, I and R) of the whole CA. Each output region model redefines the `ExtOutput` function using a different function from `epidemics.c`, in order to observe the desired variable. The simulation results after 50 steps are shown in Fig. 10.

In order to demonstrate the simulation of a larger space, this model has been simulated with an space size of 500x500. This generates around 750,000 equations, due to the matrices defined in the output region models. These matrices are managed by the Modelica simulation algorithm. Dymola generates a C file of 140MB which is difficult to compile and simulate. Because of this the output region models

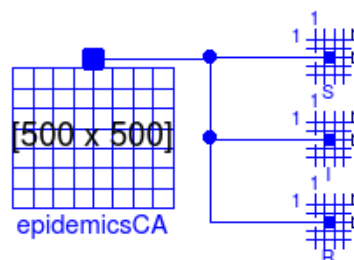


Figure 9: CA model of SIR epidemics spread.

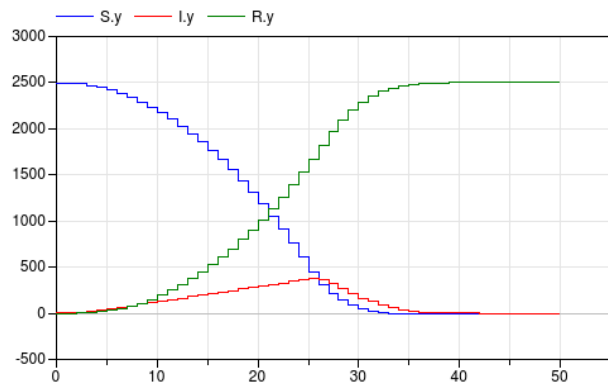


Figure 10: Evolution of the sum of the state variables (S,I and R) of the whole CA after 50 steps.

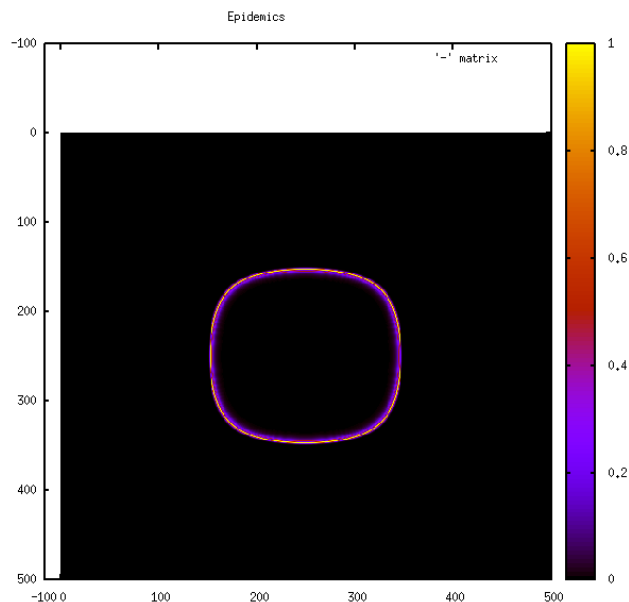


Figure 11: Capture of graphical animation at  $t = 100$  for the epidemic spread model using a 500x500 grid.

are removed from the CA model before performing the simulation. The capture of the graphical animation at the final step ( $t = 100$ ) is shown in Fig. 11.

## 6 Conclusions

A new Modelica library has been developed to facilitate the description of CA models. The simulation algorithms are transparent to the user, who has to focus on the description of the behavior of the model. The behavior of the models is described using external C functions. The use of external functions improves the performance and scalability of the simulations. The functionality of the library has been demonstrated by means of two case studies.

Some future work ideas are: to support the description of 3D models; to improve the generation of the graphical animation using graphical libraries instead of Gnuplot; to develop a graphical interface to define the initial conditions of the CA model; and to automatically parallelize the simulation of the CA in order to improve the performance.

## 7 Acknowledgments

This work has been supported by UNED, under 2013-026-UNED-PROY grant.

## References

- [1] von Neumann J. Theory of self-reproducing automata. Univ. of Illinois Press, Urbana and London, 1966.
- [2] Ilachinski A. Cellular Automata: A Discrete Universe. World Scientific, Singapore, 2001.
- [3] Schiff J.L. Cellular Automata: A Discrete View of the World. Wiley-Interscience, New York, USA, 2008.
- [4] Wolfram S. A New Kind of Science. Wolfram Media Inc., Champaign, IL, USA, 2002.
- [5] Vangheluwe H.L.M., Vansteenkiste G.C. The cellular automata formalism and its relationship to DEVS. In: In 14th European Simulation Multi-conference (ESM).
- [6] Hötendorfer H., Estelberger W., Breitenecker F., Wassertheurer S. Three-dimensional cellular automaton simulation of tumour growth in inhomogeneous oxygen environment. *Mathematical and Computer Modelling of Dynamical Systems*, 15:pp. 177–189, 2009.
- [7] O’Sullivan D., Torrens P.M. Cellular Models of Urban Systems. In: S. Bandini, T. Worsch, eds., *Theoretical and Practical Issues on Cellular Automata*. Springer-Verlag, London, 2000.
- [8] Kier L.B., Seybold P.G., Cheng C.K. Modeling Chemical Systems using Cellular Automata. Springer, Dordrecht, The Netherlands, 2005.
- [9] Rouhaud J.F. Cellular automata and consumer behaviour. *European Journal of Economic and Social Systems*, 14:pp. 37–52, 2000.
- [10] Kroc J., Sloot P.M., Hoekstra A.G., eds. *Simulating Complex Systems by Cellular Automata*. Springer-Verlag, Berlin, 2010.
- [11] Ganguly N., Sikdar B.K., Deutsch A., Canright G., Chaudhuri P.P. A Survey on Cellular Automata. Tech. rep., 2003.
- [12] Fritzson P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Computer Society Pr, 2003.
- [13] Sanz V., Urquia A. An Approach to Cellular Automata Modeling in Modelica. In: *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pp. 121–130. Nottingham, UK, 2013.
- [14] Sanz V., Urquia A., Cellier F.E., Dormido S. System Modeling Using the Parallel DEVS Formalism and the Modelica language. *Simulation Modeling Practice and Theory*, 18(7):pp. 998–1018, 2010.
- [15] Sanz V., Urquia A., Dormido S. Parallel DEVS and Process-Oriented Modeling in Modelica. In: *Proceedings of the 7<sup>th</sup> International Modelica Conference*, pp. 96–107. Como, Italy, 2009.
- [16] Zeigler B.P., Kim T.G., Prähofer H. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2000.
- [17] Sanz V. Hybrid System Modeling Using the Parallel DEVS Formalism and the Modelica Language. Ph.D. thesis, ETSI Informática, UNED, Madrid, Spain, 2010.
- [18] Zimmer D. Module-Preserving Compilation of Modelica Models. In: *Proceedings of the 7<sup>th</sup> International Modelica Conference*, pp. 880–889. Como, Italy, 2009.
- [19] Modelica Association. *Modelica - An Unified Object-Oriented Language for Physical Systems Modeling*. Language Specification version 3.3, 2013. URL <http://www.modelica.org/documents>.
- [20] Williams T., Kelley C. *Gnuplot 4.6: An Interactive Plotting Program*, 2012. URL <http://www.gnuplot.info>.
- [21] White S.H., del Rey A.M., Sanchez G.R. Modeling epidemics using cellular automata. *Applied Mathematics and Computation*, 186(2007):pp. 193–202, 2007.

# Physiolibrary - Modelica library for Physiology

Marek Mateják\*, Tomáš Kulhánek\*, Jan Šilar\*, Pavol Privitzer\*, Filip Ježek\*\*, Jiří Kofránek\*

\*Institute of Pathological Physiology, 1st Faculty of Medicine, Charles University in Prague  
U nemocnice 5, Prague 2, 128 53, Czech Republic

\*\*Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in  
Prague, Technická 2, Prague 6

marek@matfyz.cz

## Abstract

Physiolibrary is a free open-source Modelica library designed for modeling human physiology. It is accessible on the Modelica Libraries web page at <https://www.modelica.org/libraries>. This library contains basic physical laws governing human physiology, usable for cardiovascular circulation, metabolic processes, nutrient distribution, thermoregulation, gases transport, electrolyte regulation, water distribution, hormonal regulation and pharmacological regulation.

*Keywords: Physiolibrary; HumMod; Modelica library; Physiology; Integrative physiology; System biology*

## 1 Introduction

Our laboratory have a long tradition building physiological libraries, starting with the Matlab/Simulink environment [2]. The origin of this Modelica Physiolibrary was in the first version of our HumMod Golem Edition model implementation [3-6], where it was called HumMod.Library. As the successors of Guyton's Medical Physiology School write, the original HumMod model [7] is "The best, most complete, mathematical model of human physiology ever created" [8].

We are also developing many types of smaller physiological models for use in medical education [9-11], so it was essential to separate this library from our HumMod Modelica implementation. Some other Modelica models and libraries covering the biological domain already existed, e.g. [12-17], which are useful in the process of system modeling and parameter identification. Especially BioChem Modelica library, that implements large part of SBML library in Modelica language [14-18].

Our Physiolibrary contains only carefully-chosen elementary physiological laws, which are the basis of more complex physiological processes. For example from only three type of blocks (ChemicalReaction, Substance and MolarConservationMass) it is possible to compose the allosteric transitions [19] or the Michaelis-Menten equation.

## 2 Physiology

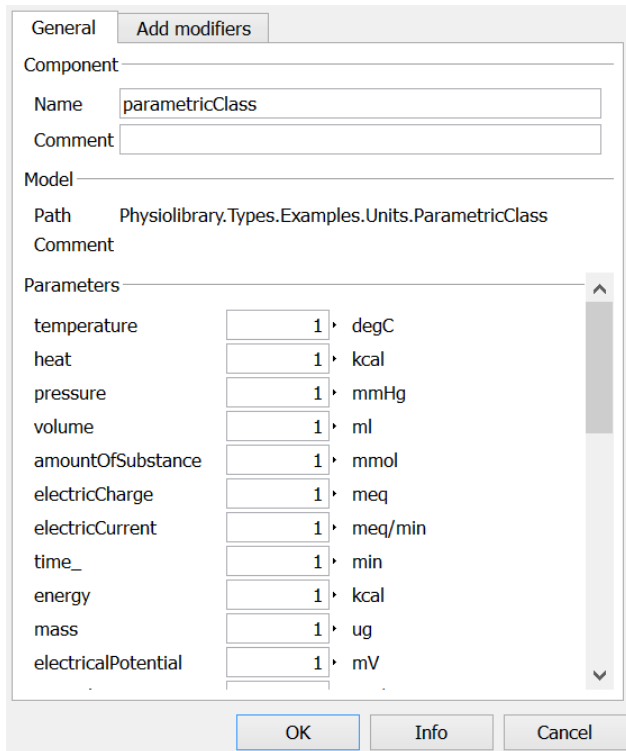
Physiology is a very progressive discipline, that examines how the living body works. And it is no surprise that all processes in the human body are driven by physical laws of nature. The great challenge is to marry old empirical experiments with the "new" physical principles. Many teams and projects in the world deal with this formalization of physiology, for example: Physiome[20], SBML[17, 18], EuroPhysiome[21], VPH[22], CellML[23] etc. It is our hope that this library helps this unflagging effort of physiologists to exactly describe the processes.

### 2.1 Display units in physiology

Energy in medicine and chemistry has a very long tradition. One must not be confused by its different units and definitions. The researcher must be aware of multiple definitions of calorie, such as the international calorie, the 15°C calorie, the thermal calorie or the Calorie with a capital "C". The origin of this unit is in the thermal energy needed to heat one gram of water by one degree Celsius. But because the measurement conditions may differ, these alternative definitions are necessary. In physiology it is recommended to use only international calorie as defined in Table 1. The flow of heat/energy is usually calculated in kcal/min, but in physics this is called power and is expressed in the SI unit watts.

Pressure units in medicine are also mainly based on historical measurements. For many years blood

pressure was measured by the mercury sphygmomanometer, where the pressure is represented by the change of mercury hydrostatic column height. And because the scale of units on the column is in millimetres the pressure unit is called millimetre of mercury 'mmHg'. There also exists a very small difference between this unit and torrs. It is caused again by variance in measurement conditions.



**Figure 1, Example parameter dialog for non-SI physiological units. The Dymola environment automatically converts this user’s non-SI-values to SI-values to ensure compatibility with any other Modelica library.**

Many physiological processes are based on electrical principles in the human body. The main cause of this is that each cell has a nonconductive membrane with molecular structures called channels, through which the fluxes of electrolytes can be precisely regulated. Even more, the cells use energy from metabolism to retain a small electric potential between inside and outside. This view leads to a unit called equivalents or “eq”. A charge of 1eq, for example, has 1mol of sodium cations (Na<sup>+</sup>). The fluxes of electrically charged ions can be in meq/min, but in physics the SI unit ampere is more generally used.

Unit conversion table (for Modelica environment display-unit setting)		
x kcal	=	4186.8*x J
x kcal/min	=	69.78*x W

x mmHg	=	133.322387415*x Pa
x degC	=	273.15 + x K
x meq	=	96.4853365*x C
x meq/min	=	1.60808894*x A
x mosm	=	0.001*x mol
x litreSTP	=	0.044031617*x mol
x litreSATP	=	0.040339548*x mol
x litreNIST	=	0.041571200*x mol

**Table 1, Selected Non-SI units in physiology**

Another strange unit describing the amount of substance is the osmol (“osm”), which has the same value as the mol, but which highlights the property that this substance cannot cross the membrane together with the flux of its solvent.

For gases, it is common to measure the amount as volume, which for specific measurement conditions is equivalent to the number of molecules. The International Union of Pure and Applied Chemistry (IUPAC) set this standard condition for temperature and pressure (STP) precisely at 0°C and 100kPa. But other standards exist. For example, SATP is measured at 25°C and 100kPa, or the standard measurement condition at the National Institute of Standards and Technology (NIST), which is 20°C and 101.325kPa.

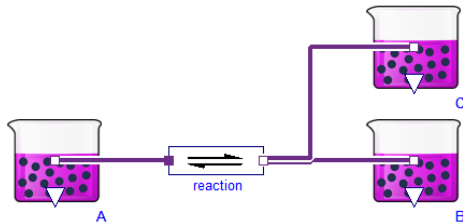
## 2.2 Chemical domains in physiology

In physiology books, chapters about chemical substances are organized by their types. The main reason for this is that each substance in the human body is regulated in a different way. For example the regulation of sodium is different from the regulation of potassium, and from the regulation of glucose, and so on. This view leads to the idea of having separate models of each substance. The origin of different flows and regulations is the (cellular) membrane. Water and solutions can cross it in different directions at the same time. Crossings occur for different reasons: water is driven mostly by osmotic gradients, electrolytes are driven by charge to reach Donnan's equilibrium, and some solutes can even be actively transported against their concentration or electrical gradients. And all this is specifically driven from the higher levels by neural and hormonal responses.

In Physiolibrary flows and fluxes of solutes are supported mostly by the Chemical package. All parts inside this Physiolibrary.Chemical package use the connector ChemicalPort, which defines the molar concentration and molar flow/flux rate of one solute. This is the supporting infrastructure for modeling membrane diffusion, accumulations of substances,

reversal chemical reactions, Henry's law of gas solubility, dilution with additional solvent flow, membrane reabsorption, chemical degradation and physiological clearance.

For usage examples, please open the `Chemical.Examples` package, where the chemical reaction shown in Fig. 2 is implemented, along with many other chemical processes.



**Figure 2, Example of a chemical reaction:  $A \leftrightarrow B + C$ .** Purple beakers (Substance) accumulate one type of substance and generate its concentration in port. Block for chemical reaction (ChemicalReaction) can have any number of substrates or products with any stoichiometric numbers. In this case there is only one substrate and two products. Purple lines represent chemical connectors, are composed of molar concentration and the molar flow of substance.

The graphically-created diagram shown in Fig. 2 generates this Modelica code:

```

model SimpleReaction2
import PhysiLibrary.Chemical.Components.*;
Substance A(solute_start=0.9);
ChemicalReaction reaction(K=1, nP=2);
Substance B(solute_start=0.1);
Substance C(solute_start=0.1);
equation
connect(A.q_out, reaction.substrates[1]);
connect(reaction.products[1], B.q_out);
connect(reaction.products[2], C.q_out);
end SimpleReaction2;

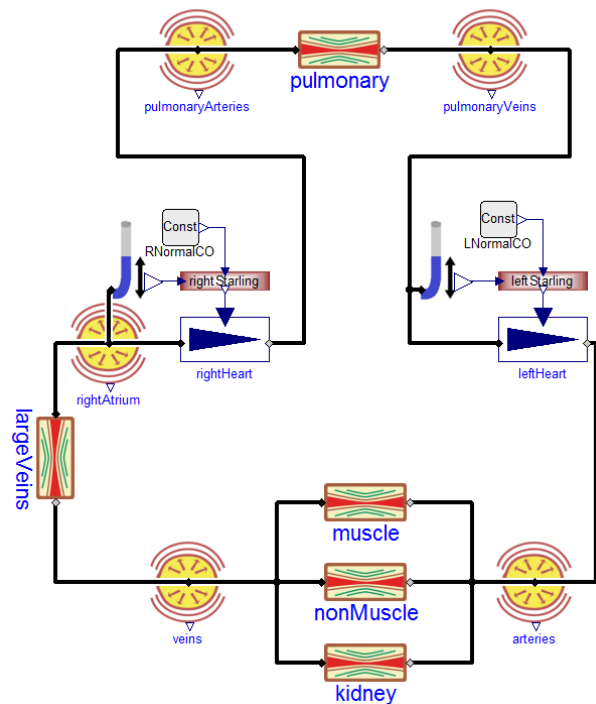
```

This means that before the numerical simulation begins, each `Substance.solute_start` parameter must be set to some initial amount of substance. `ChemicalReaction.nP` must also be configured for the number of products, and parameter `ChemicalReaction.K` must be configured for the dissociation constant of reaction in SI-units (please note that concentration of  $1 \text{ mol/m}^3 = 1 \text{ mmol/L}$ ). As mentioned before, the values in text code are in SI-units, but the Dymola environment support non-SI units in the parameter dialog of each component.

### 2.3 Hydraulic domain in physiology

The main usage of the hydraulic domain in human physiology is modeling of the cardio-vascular system. And because there are no extreme thermo-

dynamic conditions, the system can be really simple—it is only necessary to model conditions for incompressible water, at normal liquid-water temperatures and with relative pressure 5-20kPa. This boring thermodynamic state leads to using very simple blocks of hydraulic resistance, hydrostatic pressure, volumetric flow, inertia and finally a block representing blood accumulation in elastic vessels.



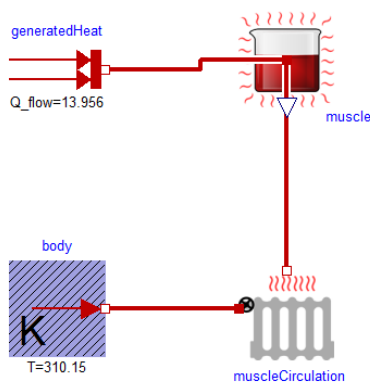
**Figure 3, Hydraulic example: the cardiovascular subsystem of the famous Guyton-Coleman-Granger model [1].** Yellow circles (ElasticVessel) represent blood accumulation and pressure generation, rectangles between them are hydraulic resistances (Resistance) of blood vessels, blue triangles (Pump) represent a heart pump driven by the Frank-Starling law. Heart-filling pressures are determined by the block with the blue tube icon (PressureMeasure) and a block-rectangle (Blocks.Factors.Spline) converts filling pressure to their effect on cardiac output. Black lines connect the hydraulic connectors (HydraulicPort), which contains pressure and volumetric flow variables.

### 2.4 Thermal domain in physiology

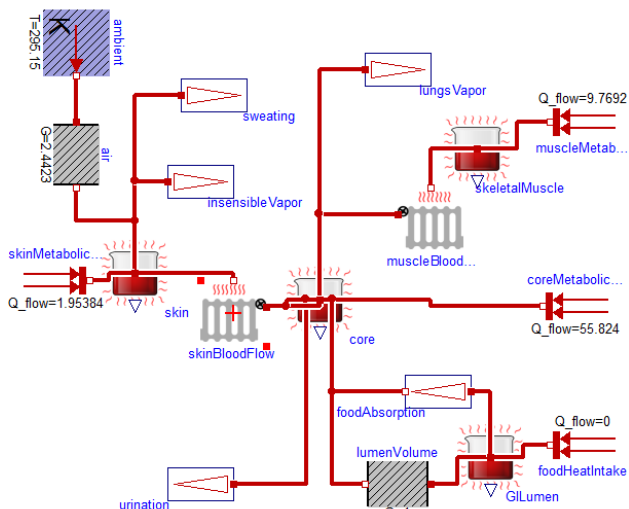
For the human body to function optimally, it is critical to hold the core temperature at  $35\text{--}39^\circ\text{C}$ . A fever of  $41^\circ\text{C}$  for more than a short period of time causes brain damage. If the core temperature falls below  $10^\circ\text{C}$ , the heart stops. As in the hydraulic domain, the thermal domain is simplified to these conditions.

The `PhysiLibrary.Thermal` package extends the package `Modelica.Thermal.HeatTransfer` from

Modelica Standard Library 3.2 (MSL), where the connector is composed of temperature and heat flow. The main blocks in `Physiolibrary.Thermal` are: `Conductor`, `IdealRadiator` and `HeatAccumulation`. The heat conductor conducts the heat from the source, such as muscles or metabolically active tissue, to its surrounding. `IdealRadiator` delivers heat to tissues by blood circulation. `HeatAccumulation` plays a role in accumulating thermal energy in each tissue mass driven by its heat capacity. We recommend using this block instead of `Modelica.Thermal.HeatTransfer.HeatCapacitor` to allow the possibility of variable mass amounts, and to have support for calculating steady state, described in section 2.7.



**Figure 4, Example of heat flow from a working muscle.** The muscle is represented by a red beaker (`HeatAccumulation`), where heat energy is accumulated in a mass with defined weight and specific heat. Heat transfer is processed by blood circulation (`IdealRadiator`) with blood flow as its internal parameter. The temperature of blood is set to a fixed value of 37°C to simulate well-regulated core body temperature.



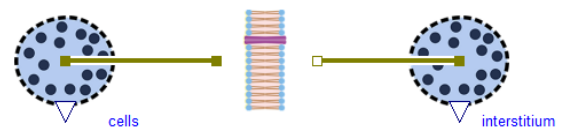
**Figure 5, Basic heat flow model of human body.** Heat production occurs in each tissue via metabolism, or from warm (to cool) eaten food using the MSL block

with two red arrows (`Modelica.Thermal.HeatTransfer.Sources.FixedHeatFlow`). This heat is stored in tissues (`HeatAccumulation`) and transferred out by blood (`IdealRadiator`) or together with mass (`Stream`, `HeatOutstream`), where the model also integrates vaporization heat loss. Heat radiation and conduction to the environment is simplified using an MSL block for heat conductor (`Modelica.Thermal.HeatTransfer.Components.ThermalConductor`).

## 2.5 Osmotic domain in physiology

One of the basic phenomenon of biological systems is the osmotically-driven flow of water. This is always connected with semipermeable membranes. The different concentrations of impermeable solutes on both sides of the membrane causes the hydrostatic pressure at the concentrated side to rise[24]. This pressure difference is called osmotic pressure. Osmotic pressure is linearly proportional to the concentration gradient of impermeable solutes. The osmolarity (osmotic concentration) is also one of the main indexes of human body balance, called homeostasis. Its value should not significantly deviate for a long period of time from a value of 285-295mosm/l.

In `Physiolibrary` the osmotic connector `OsmoticPort` is composed of the osmotic concentration and the volumetric flux of permeable liquid. The two main blocks are called `Membrane` and `OsmoticCell`. Here, inside the membrane blocks, it is of course possible to also define hydraulic pressure and temperatures effects on both sides of membrane.



**Figure 6, Osmotic example simulating water transfer between intracellular and interstitial compartments in hypertonic or hypotonic conditions.**

## 2.6 Types and units

The most common errors in `HumMod Golem Edition` were caused by using bad physical units. The main problem of medical research, articles, and experiments is using obscure units from medicine, pharmacology, biology and non-physics disciplines.

The `Physiolibrary` fulfills the `Modelica` ideal of using SI units as the main unit for each variable, and the previously described physiological units are also implemented as the `displayUnits` for each variable. Using these `displayUnits` the user sets and sees the "physiological" values. The implementation can also be joined to any unit-correct `Modelica` models and physical equations without crashing due to unit in-

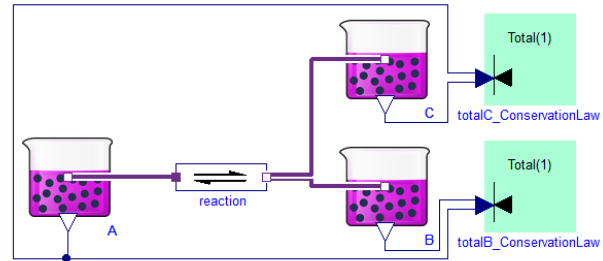
compatibilities. The unit support of Physiobrary is so strong that one can even chose the right unit-typed "input real"/"output real" from the library package `Types.RealIO` or one can use unit-typed constants (`Types.Constants`). As can be expected, only the non-specific package Blocks in the Physiobrary has variables without units.

## 2.7 Steady states

One of the main questions in clinical medicine is how to stabilize the patient. In modeling the oscillating heart, breathing, circadian rhythm or menstruation cycle the model can be designed as non-oscillating, using variables such as period times, amplitudes, frequencies, mean values and other phase space variables. This type of model has better numerical stability for long simulation times, and even more importantly, it can be "stabilized". This stabilization we call a steady state.

To be mathematically exact, we define a *steady state system* (SSS) as a non-differential system derived from an original differential system (DS) by using zero derivations and by adding additional steady state equations (ASSE). The number of the ASSE must be the same as the number of algebraically dependent equations in the non-differential system derived from DS by setting zero derivations. The ASSE describes the system mostly from the top view, such as the equations of mass conservation laws or the boundary equation of environment sources.

To define a model as an SSS, the user must switch each Simulation parameter in each block to the value `Types.SimulationType.SteadyState` and must have correctly defined all necessary ASSE. This setting causes the system to ignore any start values for any state and add zero derivation equations instead. There does not currently exist a Modelica environment which can automatically find and remove generated dependent equations by this solution. So the correct number of states must be marked as dependent (parameter `isDependent`) and the same number of ASSE must be inserted. However, despite the fact that a model in this steady-state setting will not be locally balanced, it must be globally balanced and without any dependent equation.



**Figure 7, Steady state system example: equilibrium of chemical reaction  $A \leftrightarrow B + C$  is calculated with two ASSE. Zero change of reactant A is automatically propagated through a reaction to both products. So both products must be marked as dependent (parameter `isDependent`), and two mass conservation laws must be added as green square blocks to the right (`MolarConservationLaw`). Please note that conservation laws must be included only after designing the rest of the system, because they are global properties, not properties of individual substances or reactions.**

Adding an ASSE is possible by inserting and connecting the energy or mass conservation law block from package `SteadyState.Components`. Another possibility is using environment sources blocks, where setting the `isIsolatedInSteadyState` parameter adds the equation of the zero mass/volume/energy flow from or to environment.

The steady state model often changes to one big nonlinear strong component, but without solver stiff or convergence problems. Especially in quick chemical kinetics, it is not necessary to have very rapid molar fluxes, when it always reach equilibrium in the end. This design is also useful in creating steady-stated parts in a dynamical model without huge rebuilding. It also brings other benefits. To see these possibilities, one has to realize that conservation laws could be invariances in a dynamical simulation. This is really useful for debugging.

For example see the model `SteadyStates.Examples.SimpleReaction2_in_Equilibrium` (Fig.7), which implements the equilibrium of the closed system from Fig.2 as a solution of three chemical substances with a simple reversible reaction between them extended by two conservation laws. Each of these laws describes the total possible amount of one product in its free form and in its associated form.

It is always a big challenge to nicely solve initial values of differential system. However, it should be possible to solve the SSS in its initial phase. And this is the idea behind the `Types.SimulationType.InitSteadyState` option for models already extended with ASSE.

## 2.8 File utilities—input/output manipulations

During the creation and debugging of huge integrated models it is necessary to easily define consistent input, output and test sets of all output variables for some subsystems. Let's imagine that we have a model composed only of subsystems that converge from some constant inputs to constant outputs. It should be possible to substitute each main subsystem for its chosen constant output values as parameters. Comparing the model with these parametric values and the original subsystem can show the wrong part of the simulation.

For example in the huge HumMod model it is necessary to debug smaller parts separately. These tools could be use, because HumMod is the type of constant-converged model. Each subsystem in the first level has the constant input values set for its output variables. Simulating, for example, the cardiovascular subsystem is possible by creating the high-level system with the original cardiovascular subsystem, but with a constant metabolic, constant thermoregulation, constant hormonal, constant water, constant proteins, constant gases, constant electrolytes and constant status subsystem.

Because the number of output variables for each subsystem changes during development, it is a good idea to have only one list for each subsystem. And generating consistent sets to store, restore, compare initial and final values is possible by the same pattern as presented in the package Types.Example. In this package it is also possible to define a customized way to save and load the variables that connect subsystems together. For this purpose, one has to redeclare the package Types.Utilities with simple functions for reading and writing values, such as is done in the default package FileUtilities.

The typical code of a parameter set could be:

```
model MyParameterSet
  replaceable package T = Physiolibary.Types.RealTypes
    constrainedby Physiolibary.Types.RealTypes;

  T.Pressure      v1(varName="Bone-Flow.PO2");
  T.VolumeFlowRate v2(varName="Bone-Flow.BloodFlow");
  T.MolarFlowRate v3(varName="Bone-Metabolism.O2-Need");
  T.Volume        v4(varName="Bone-Tissue.LiquidVol");

  BusConnector busConnector;

equation
  connect(v1.y, busConnector.Bone_PO2);
  connect(v2.y, busConnector.Bone_BloodFlow);
  connect(v3.y, busConnector.Bone_O2Need);
  connect(v4.y, busConnector.Bone_LiquidVol);
end MyParameterSet;
```

To redefine these sets of values to use inputs from a file is simple. The user just redeclares the type of the values to type InputParameter and redirects reading functions to FileUtilities:

```
model InputParameterSet
  extends MyParameterSet( T(redeclare block Variable =
    Physiolibary.Types.RealExtension.InputParameter (
      redeclare package Utilities = Physiolibary.Types.FileUtilities));
  end InputParameterSet;
```

And the same set of values can also be redefined to file output at the end of the simulation:

```
model OutputFinalSet
  extends MyParameterSet( T(redeclare block Variable =
    Physiolibary.Types.RealExtension.OutputFinal (
      redeclare package Utilities = Physiolibary.Types.FileUtilities));
  end OutputFinalSet;
```

## 3 Conclusion

In our opinion the best way to understand this library is to download it from the Modelica web pages at [www.modelica.org/libraries](http://www.modelica.org/libraries) and examine the examples. We recommend examining the package *Hydraulica.Examples*, which provides an example of a simplified cardiovascular system; the package *Chemical.Examples*, which provides an example of allosteric hemoglobin oxygen binding; the package *Osmotic.Examples*, which simulates cell volume in hypertonic and hypotonic environments; and finally the package *Thermal.Examples*, which simulates the heating of circulated blood inside active muscles.

## Acknowledgements

This paper describes the outcome of research that has been accomplished as part of a research program funded by the Ministry of Industry and Trade of the Czech Republic by the grant FR—TI3/869 and by The Ministry of Education, Youth and Sports by the grant SVV-2013-266509.

In addition, the main author wants to thank Jan Obdržálek for consultations about physical units, and Austin Schaefer for English language editing.

## References

- [1] Guyton, A.C., T.G. Coleman, and H.J. Granger, *Circulation: overall regulation*. Annual review of physiology, 1972. **34**(1): p. 13-44.



- [2] Teaching, L.o.B.a.C.A. *Physiolibrary in Matlab and Simuling*. 2008; Available from: <http://www.physiome.cz/simchips>.
- [3] Marek Matejak and J. Kofranek, *HUMMOD–GOLEM EDITION–ROZSAHLY MODEL FYZIOLOGICKYCH SYSTEMU* Medsoft, 2011: p. 182-196.
- [4] Marek Matejak and J. Kofranek, *Rozsahly model fyziologickych regulacı v Modelice*. Medsoft, 2010: p. 126-146.
- [5] Kofranek, J., M. Matejak, and P. Privitzer. *HumMod - large scale physiological model in Modelica*. in *8th. International Modelica conference*. 2011. Dresden, Germany.
- [6] Kofranek, J., Matejak, M., Privitzer, P., Tribula, M., Kulhanek, T., Silar, J., Pecinovsky, R. *HumMod-Golem Edition: large scale model of integrative physiology for virtual patient simulators*. in *World Congress in Computer Science 2013 (WORLDCOMP'13), International Conference on Modeling, Simulation and Visualisation Methods (MSV'13)*. 2013.
- [7] Hester, R.L., et al., *HumMod: a modeling environment for the simulation of integrative human physiology*. *Frontiers in physiology*, 2011. **2**.
- [8] Center, L.b.U.o.M.M., *HumMod*. 2012: p. <http://hummod.org/>.
- [9] Kofranek, J., M. Matejak, and P. Privitzer. *Leaving toil to machines - building simulation kernel of educational software in modern software environments*. in *Mefanet 2009*. 2009. Masaryk University, Brno.
- [10] Kofranek, J., M. Matejak, and P. Privitzer, *Web simulator creation technology*. MEFANET report, 2010. **3**: p. 52-97.
- [11] Kofranek, J., et al., *The Atlas of Physiology and Pathophysiology: Web-based multimedia enabled interactive simulations*. *Computer methods and programs in biomedicine*, 2011. **104**(2): p. 143-153.
- [12] Pro, S. and B. Bachmann, *An Advanced Environment for Hybrid Modeling and Parameter Identification of Biological Systems*.
- [13] Cellier, F.E. and A. Nebot. *Object-oriented Modeling in the Service of Medicine*. in *Proc. 6th Asia Simulation Conference*. 2005.
- [14] Larsdotter Nilsson, E. and P. Fritzson. *BioChem-A Biological and Chemical Library for Modelica*. in *Proceedings of the 3rd International Modelica Conference (November 3-4, Linkoping, Sweden)*. 2003. Modelica Association.
- [15] Nilsson, E.L. and P. Fritzson, *Biochemical and metabolic modeling and simulation with Modelica*. BioMedSim. Linkoping, Sweden, 2005.
- [16] Nilsson, E.L. and P. Fritzson. *A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems*. in *Paper presented at the 4th International Modelica Conference*. 2005.
- [17] Brugard, J., et al. *Creating a Bridge between Modelica and the Systems Biology Community*. in *7th International Modelica Conference, Como, Italy*. 2009.
- [18] Hucka, M., et al., *The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models*. *Bioinformatics*, 2003. **19**(4): p. 524-531.
- [19] Monod, J., J. Wyman, and J.-P. Changeux, *On the nature of allosteric transitions: a plausible model*. *Journal of Molecular Biology*, 1965. **12**(1): p. 88-118.
- [20] Bassingthwaighe, J.B., *Strategies for the physiome project*. *Annals of Biomedical Engineering*, 2000. **28**(8): p. 1043-1058.
- [21] Fenner, J.W., et al., *The EuroPhysiome, STEP and a roadmap for the virtual physiological human*. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2008. **366**(1878): p. 2979-2999.
- [22] Hunter, P.J. and M. Viceconti, *The VPH-physiome project: standards and tools for multiscale modeling in clinical applications*. *Biomedical Engineering, IEEE Reviews in*, 2009. **2**: p. 40-53.
- [23] Smith, L., et al., *SBML and CellML translation in Antimony and JSim*. *Bioinformatics*, 2013: p. btt641.
- [24] Mortimer, R.G., *Mathematics for physical chemistry*. 1999: Access Online via Elsevier.



# Modelling of Electrical Power Systems with Dynamic Phasors in Modelica

Tao Yang Serhiy Bozhko Greg Asher  
University of Nottingham  
University Park, Nottingham, NG7 2RD, UK  
{Tao.Yang, Serhiy.Bozhko, Greg.Asher}@nottingham.ac.uk

## Abstract

The more-electric aircraft has been identified as the dominant trend for the future aircraft. With the development of power electronics, advanced control and drive systems, a great increase number of new electrical loads will be seen on-board. This paper presents an electrical power system (EPS) library developed in Modelica based on the dynamic phasor concept. The developed library uses a modular modelling concept. This makes the library more flexible and user-friendly. The application of developed library is also demonstrated through simulations of a MOET-architecture EPS.

*Keywords: Dynamic Phasor; More-Electric Aircraft; Modelica*

## 1 Introduction

The more-electric aircraft (MEA) has been identified as a dominant trend for the next-generation aircraft. The recent advance of power electronics, electrical drives and modern control theory makes it possible to replace many functions, which are conventionally managed by hydraulic, pneumatic and mechanical power, with electrical power driven devices [1]. This move offers reduced overall system weight, as well as increased efficiency, reliability and performance of the aircraft. The future EPS may take many forms: AC, DC, hybrid, frequency-wild, variable voltage, together with the possibility of novel connectivity topologies. To address the stability, availability and capability issues as well as to assess the performance of the power quality and transient behaviour, extensive simulation work is required to develop the EPS architectures.

Due to the switching behaviour of power electronic devices and its resulted higher harmonics, it is very

time-consuming and even impractical to simulate a large-scale EPS with some non-linear and time-varying models. Considering the system dynamic frequency is normally much lower than that of higher harmonics, it is a common practice to neglect these higher harmonics when studying system dynamic behaviour. The average modelling technique, which removes the higher harmonics by averaging the variable during one fundamental period, has been widely used to model the EPS recently. In these average models, the variables are transformed from the three-phase *abc* frame to a synchronous rotating *dq* frame (DQ0 models). The DQ0 model demonstrates good performance and high efficiency under balanced conditions [2]. This model, however, becomes very slow under unbalanced or faulty conditions. This is due to the second harmonic present in the DQ0 model under unbalanced conditions.

In this paper, a general averaging modelling technique, referred to as dynamic phasors (DP), is introduced and a model library based on this concept is developed in Modelica. The DPs are in nature some time-varying Fourier series coefficients. Compared with conventional phasors, the DP can model the system even it is not in the steady state. Truncating unimportant higher order harmonics and only considering the significant components, DP models are capable of retaining the dominant dynamic features of the EPS and suitable for transient performance studies. The slow variation of DPs allows for larger simulation time steps and results in faster simulations under both balanced and unbalanced conditions.

The software used to develop models is Dymola, standing for dynamic modelling laboratory. This software uses the open Modelica modelling language which allows users to freely create their own model libraries or modify the ready-made model libraries. This modelling language has been widely

used in modelling electrical power systems [3, 4]. In contrast to data flow-oriented languages with directed inputs and outputs, such as the widely known Matlab/Simulink tool, Modelica employs an equation-based modelling technique and all the variables are treated equally. This avoids the algebraic loop issues frequently occurred in Matlab/Simulink. The equation-based modelling concept also results in a faster modelling process and a significantly increased re-usability, since the interconnection between models is easier and simpler than that of the signal-flow based modelling. There is no need to explicitly define the interface equations and predefine the input and output signals.

In this paper, the dynamic phasor concept is briefly introduced. The development of DP models in Modelica is explained using an RLC circuit. DP models of key EPS elements, such as generators, PWM converters etc. are introduced. The application of the DP library is demonstrated through simulation of the MOET architecture EPS.

## 2 Dynamic Phasors

Before introducing the developed library, a brief introduction of the dynamic phasor concept is introduced in this section. The DPs essentially are some time-varying Fourier coefficients. For a time-domain quasi-periodic waveform  $x(\tau)$ , defining a time-moving window  $\tau \in (t-T, t]$ , as shown in Figure 1, and viewing the waveforms in this window to be periodic, the Fourier expansion of the waveform in this interval can be represented by the following Fourier series [5]:

$$x(\tau) = \sum_{k=-\infty}^{\infty} X_k(t) e^{jk\omega_s \tau} \quad (1)$$

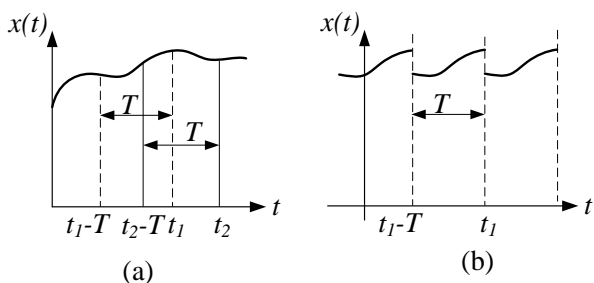


Figure 1 (a) Defined moving window at time  $t_1$  and  $t_2$ , (b) equivalent periodic signal at time  $t_1$

where  $\omega_s = 2\pi/T$  and  $T$  is the length of the window. Though the window length  $T$  can be an arbitrary value, it is common that the fundamental period of the signal is chosen to avoid the DPs spreading over the entire frequency axis.  $X_k(t)$  is the  $k$ th Fourier coefficient in a complex form and is referred to as a “dynamic phasor”. It is defined as follows:

$$X_k(t) = \frac{1}{T} \int_{t-T}^t x(\tau) e^{-jk\omega_s \tau} d\tau = \langle x \rangle_k \quad (2)$$

where  $k$  can be any integer and is called the DP index. The triangular pair ‘ $\langle \rangle$ ’ is used as the DP calculation symbols for any time-domain variables. In contrast to the traditional Fourier coefficients, these Fourier coefficients are time-varying as the integration interval (window) slides with time. The selected set of DPs, or  $K$  with  $k \in K$ , defines the approximation accuracy of the waveform. For example, for DC-like variables and signals the index set only includes the component  $k=0$ , and for purely sinusoidal waveforms, with the window length equal to one period gives  $k=1$ .

A key factor in developing DP models is the relation between the derivatives of the variable  $x(t)$  and the derivatives of  $k$ th Fourier coefficients given as:

$$\left\langle \frac{dx}{dt} \right\rangle_k = \frac{d\langle x \rangle_k}{dt} + jk\omega_s \langle x \rangle_k \quad (3)$$

It is important to notice that, the derivative term on the right side of (3) allows the DP to study the electromagnetic transients of the power system, not limited to steady-state studies. If we drop this derivative term and fix the DP index to be  $k=1$ , the DP will reduce to the traditional phasors, which is widely used to study steady-state or quasi-steady power systems.

## 3 Modelica Library

In this session, the DP library in Modelica will be introduced. An RLC circuit, which is used to represent the transmission line in many cases, has been chosen to illustrate the DP technique. DP models of other essential elements are detailed in our previous publications [6-10] and will be briefly reviewed in this session as well.

### 3.1 DP model of RLC circuits

The DP models of RLC components are based on their time-domain voltage dynamic equations. Using the DP definition and properties, the DP transformation can be achieved conveniently.

#### (i) Resistance element

The time-domain voltage equation for a resistor can be expressed by:

$$v = Ri \quad (4)$$

If the resistance  $R$  is constant and time invariant, it can be moved out from the integration symbol and we obtain the DP form as:

$$\langle v \rangle_k = R \langle i \rangle_k \quad (5)$$

In our model, the DP index is chosen at  $K=\{0,1\}$ , the DP model in Modelica is shown below:

```

model Resistor
  extends interface.Twopins;
  parameter Modelica.SIunits.Resistance R(start=1);
equation
  v0=R*i0;
  v1.re=R*i1.re;
  v1.im=R*i1.im;
end Resistor;
    
```

where  $v0$  and  $v1$  are corresponding to DPs  $\langle v \rangle_{0,1}$ .

#### (ii) Inductance element

The time-domain voltage equation for an inductor is written as:

$$v = L \frac{di}{dt} \quad (6)$$

The DP form of (5) can be obtained by employing the DP definition and its differential as:

$$\langle v \rangle_k = \left\langle L \frac{di}{dt} \right\rangle_k = L \frac{d\langle i \rangle_k}{dt} + j\omega L \langle i \rangle_k \quad (7)$$

In Modelica, the DP model for inductors is written as:

```

model Inductor
  extends interface.Twopins;
  parameter Modelica.SIunits.Inductance L=0.001;
  Real w;
  parameter Real f=50;
equation
  w=2*Modelica.Constants.pi*f;
  v0=L*der(i0);
  v1.re=L*der(i1.re)-w*L*i1.im;
  v1.im=L*der(i1.im)+L*w*i1.re;
end Inductor;
    
```

#### (iii) Capacitance element

The DP model for a capacitor can be derived the same way as that for the inductor and is written as:

$$\langle i \rangle_k = C \frac{d\langle v \rangle_k}{dt} + j\omega C \langle v \rangle_k \quad (8)$$

```

model Capacitor
  extends interface.Twopins;
  parameter Modelica.SIunits.Capacitance C(start=2e-7);
  parameter Real f(start=50);
  Real w;
equation
  w=2*Modelica.Constants.pi*f;
  i0=C*der(v0)+0*w*C*v0;
  i1.re=C*der(v1.re)-w*C*v1.im;
  i1.im=C*der(v1.im)+w*C*v1.re;
end Capacitor;
    
```

The application of the DP models in Modelica is very convenient. In reality, it even looks the same as the models in Modelica/Dymola standard libraries. A simple RLC circuit with all the elements modelled in DPs is shown in Figure 2.

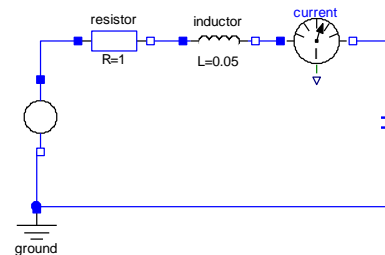


Figure 2 DP represented RLC circuit in Modelica

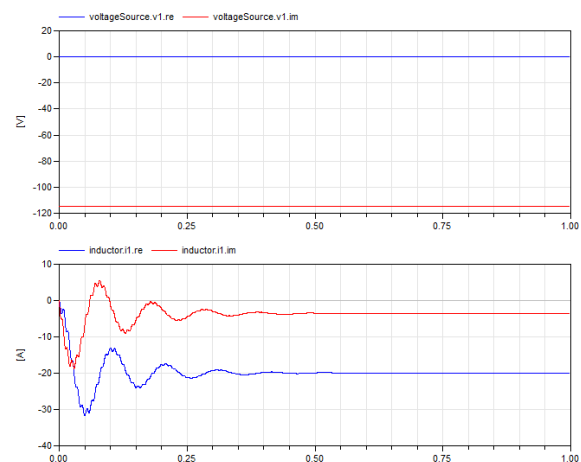


Figure 3 Simulation results of DP-modelled RLC circuit

The simulation results of the circuit are shown below. As can be seen, the sinusoidal source in DP domain becomes complex constant DC source. The currents flowing through the inductor turn into constant in the DP model.

### 3.2 DP models of synchronous generators and control

The modelling of synchronous machines has been an important topic in power system engineering for many decades. Today there are a large number of different models used in different studies. The three-stage generator system in a conventional aircraft is simplified into a synchronous generator (SG). Figure 4 showed the SG with its controlling structure.

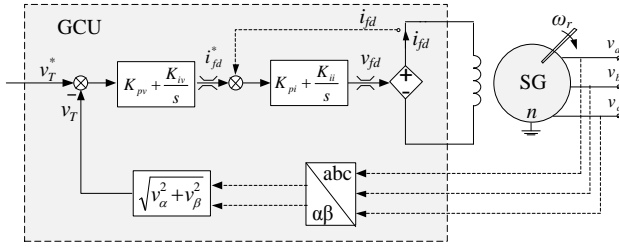


Figure 4 The equivalent circuit for the generating system in aircraft

The DP model of these machines is based on their time-domain voltage and flux equations. The fully detailed DP model of a synchronous machine has been can be found in [6] and will briefly introduced here. In DP domain, the synchronous machine can be modelled as:

$$\langle \mathbf{v}_{abc} \rangle_k = -\mathbf{r}_s \langle \mathbf{i}_{abc} \rangle_k + \frac{d\langle \boldsymbol{\lambda}_{abc} \rangle_k}{dt} + j\omega \langle \boldsymbol{\lambda}_{abc} \rangle_k \quad (9)$$

$$\langle \mathbf{v}_{dqr} \rangle_k = \mathbf{r}_r \langle \mathbf{i}_{dqr} \rangle_k + \frac{d\langle \boldsymbol{\lambda}_{dqr} \rangle_k}{dt} + j\omega \langle \boldsymbol{\lambda}_{dqr} \rangle_k \quad (10)$$

$$\langle \boldsymbol{\lambda}_{abc} \rangle_k = -\sum_m \langle \mathbf{L}_s \rangle_m \langle \mathbf{i}_{abc} \rangle_{k-m} + \sum_n \langle \mathbf{L}_{sr} \rangle_n \langle \mathbf{i}_{dqr} \rangle_{k-n} \quad (11)$$

$$\langle \boldsymbol{\lambda}_{dqr} \rangle_k = \sum_m \langle \mathbf{L}_{rs} \rangle_m \langle \mathbf{i}_{abc} \rangle_{k-m} + \sum_n \langle \mathbf{L}_r \rangle_n \langle \mathbf{i}_{dqr} \rangle_{k-n} \quad (12)$$

where  $\mathbf{L}_{ss}$  denotes the stator self-inductance matrix,  $\mathbf{L}_{rr}$  is the rotor self-inductance,  $\mathbf{L}_{sr}$  and  $\mathbf{L}_{rs}$  are the mutual inductance. The matrix  $\mathbf{r}_s$  is the resistance of stator windings and  $\mathbf{r}_r$  is the resistance of rotor windings.

The proportional-integral (PI) controllers can be converted into the DP domain with their state-space equations:

$$\dot{x} = k_i u \quad (13)$$

$$y = k_p u + x \quad (14)$$

where  $u$  is the input,  $x$  is the state variable, and  $k_p$  and  $k_i$  are the proportional and integral gains correspondingly. This equation can be converted into dynamic phasors as:

$$\frac{d\langle x \rangle_k}{dt} = k_i \langle u \rangle_k - jk\omega \langle x \rangle_k \quad (15)$$

$$\langle y \rangle_k = k_p \langle u \rangle_k + \langle x \rangle_k \quad (16)$$

The model of PI controller in modelica is written as:

```

model PI_DP
  Modelica.ComplexBlocks.Interfaces.ComplexInput u[2];
  Modelica.ComplexBlocks.Interfaces.ComplexOutput y[2];
  parameter Real kp;
  parameter Real ki;
  constant Real PI=Modelica.Constants.pi;
  Real w;
  Complex x[2];
  Real xre;

```

```

equation
  w=2*PI*f;
  //dc component
  der(xre)=ki*u[1].re;
  xre=x[1].re;
  x[1].im=0;
  y[1].re=kp*u[1].re+x[1].re;
  y[1].im=0;
  //second harmonic
  der(x[2].re)=ki*u[2].re+2*w*x[2].im;
  der(x[2].im)=ki*u[2].im-2*w*x[2].re;
  y[2].re=kp*u[2].re+x[2].re;
  y[2].im=kp*u[2].im+x[2].im;
end PI_DP;

```

### 3.3 Auto-transformer rectifier units

The multi-pulse rectifier unit is widely used to supply the HVDC bus in aircraft electrical power systems. The DP modelling of an 18-pulse auto-transformer rectifier unit (ATRU) has been discussed in our previous publication [11]. The symmetry of the ATRU is used to simplify the modelling process. The DP index for variables at the AC side terminals are set at  $K=\{1\}$ . For the DC terminal variables, the DP index is set at  $K=\{0\}$  which is equal to their time-domain values. The DP model of ATRU is shown in Figure 5 and Figure 6.

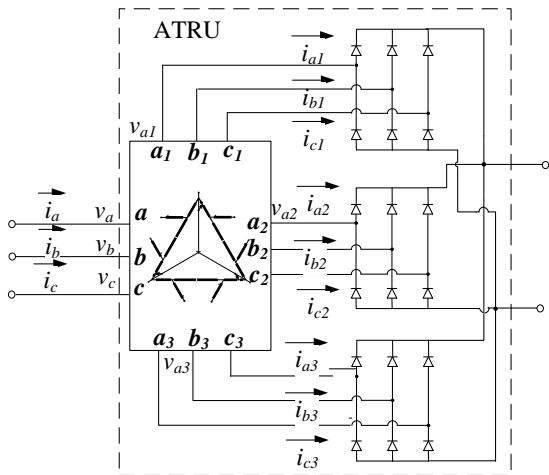


Figure 5 configuration of an 18-pulse ATRU

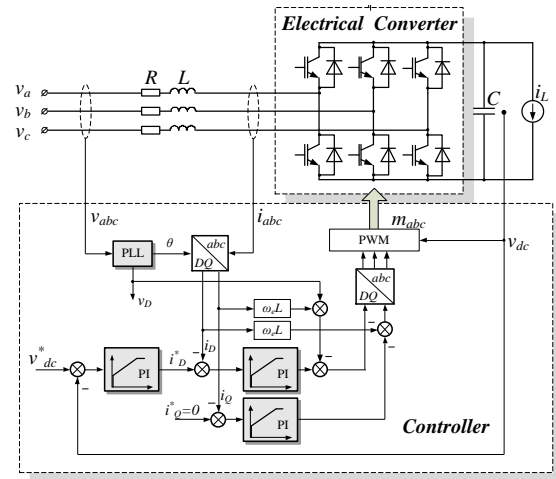


Figure 7 Structure of the PWM controlled rectifier

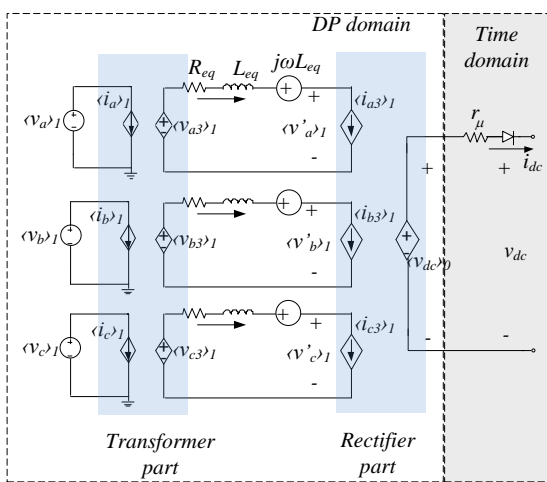


Figure 6 Equivalent circuit representation of DP model of an 18-pulse ATRU

### 3.4 Controlled rectifier units

The PWM controlled rectifier unit is well-known from previous publications and the topology is shown in Figure 7. With the voltage vector aligned with the  $d$  axis in a synchronous rotating frame, denoted as the  $DQ$  frame, the projections of the current vector onto the  $D$  and  $Q$  axes correspond to the active power and reactive power components respectively. This allows independent control of the active and reactive power flow. The electrical converter of the CRU is represented with the voltage and current relations in DP forms. The vector control of the CRU is transformed into DPs with the same strategy used in the controlled synchronous machines. The detailed development of the DP model for CRUs has been introduced in our previous publication [7]. component is considered. Thus, the DP index is set at  $K=\{0\}$ .

## 4 System Simulation

This simulation scheme has been shown in Figure 8. This EPS structure is based on the MOET large aircraft EPS architecture from Airbus France (document WP3.11 architecture V0 [12]). In the simulation studies, the electrical frequency of the SG1 is fixed at 400Hz and SG2 fixed at 405Hz. The HVAC bus voltages are controlled at 230V RMS. The power rating of elements in the system is shown in Table 1.

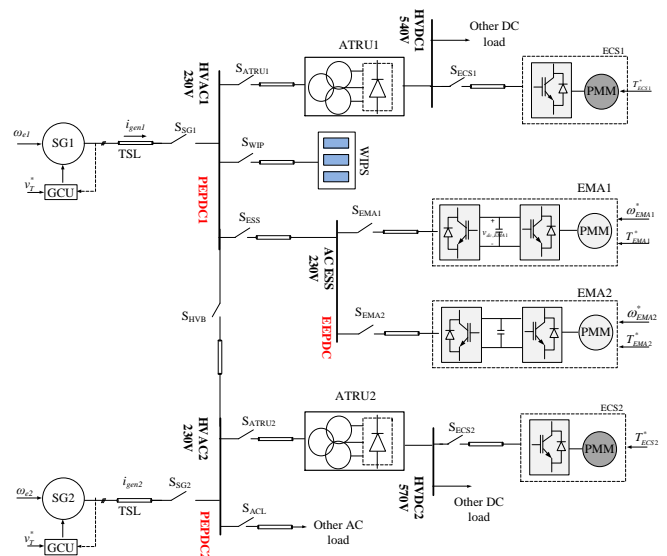


Figure 8 The EPS of the twin-generator aircraft

Table 1 Rated power of equipment in the example EPS

Equipment	Power rating
ATRU	150kW
WIPS	60kW
ECS	30kW
EMA	5kW
AC essential bus fed CRU	10kW
HVAC2 bus fed resistance	9kW
HVDC bus fed resistance	5kW

Table 2 Simulation scenarios of twin-generator aircraft EPS under normal operation conditions

Time (s)	Events
0.00	Simulation starts; EPS starts up
0.15	Switches $S_{SG1}$ and $S_{SG2}$ are closed; Switches $S_{ATRU1}$ and $S_{ATRU2}$ are closed; DC loads on HVDC buses are connected
0.20	ESC1 and ECS2 start to accelerate to the rated speed (3000rpm)
0.50	Rated load torques applied to ECS1 and ECS2 (95Nm)
0.70	WIPS changes from 60kW to 6kW
0.80	EMA1 and EMA2 start to accelerate to the rated speed (900rpm)
0.90	Rated load torque applied to EMA1 and EMA2 (54Nm)
1.00	SG1 and SG2 are connected
1.02	$S_{SG1}$ opens; SG1 removes from the system and SG2 covers the whole load system
1.20	Simulation ends

The simulation starts at  $t=0s$ . The GCU starts to regulate the HVAC bus phase voltage to 230Vrms. After the HVAC bus voltage reaches the steady state, the switches  $S_{ATRU1}$  and  $S_{ATRU2}$  are closed at  $t=0.15s$ . The SG1 starts to supply ATRU1 through HVAC1 bus. At the same time, the SG2 starts to supply ATRU2 through HVAC2 bus. At  $t=0.2s$ , the speed reference for two ECS drive system is set at 3000rpm and the rated load torque 95Nm applied to these ECS systems at  $t=0.5s$ . At  $t=0.7s$ , the

de-icing system starts to run at rated power and the WIPS is set at 60kW. After 50ms second, the de-icing process finishes and the power requirement of WIPS is reduced to 6kW to maintain the temperature of the aircraft wings. The DC-link voltage reference of EMA is set to 800V at  $t=0.6s$ . The speed reference of the EMA is set to rated speed 900rpm at  $t=0.8s$  with rated load applied at  $t=0.9s$ . In order to demonstrate parallel operation of the two generators,  $S_{HVB}$  is closed at  $t=1.0s$ . The two generators start to work in parallel for a short period, 20ms, then  $S_{HVB}$  opens and the two generators work separately again. The event sequence of start-up of the twin-generator aircraft EPS is also shown in Table 2.

Results from the ABC model (the model as a benchmark, in the three-phase coordination with switching behaviour) and DP models (all the elements are modelled in DPs) are compared in the following figures. The dynamic responses of ATRU-fed HVDC bus voltages,  $v_{HVDC1}$  and  $v_{HVDC2}$  are shown in Figure 9. The initial values of  $v_{HVDC1}$  and  $v_{HVDC2}$  are set at zero. At  $t=0.15s$ , when the switches  $S_{SG1}$  and  $S_{SG2}$ ,  $S_{ATRU1}$ ,  $S_{ATRU2}$  are closed, the inrush current push the DC-link voltage of ATRUs  $v_{HVDC1}$  and  $v_{HVDC2}$  from 0V to around 800V. From  $t=0.2s$ , the ECS starts to speed up and draws power from the generator. A slope voltage drop at HVDC bus can be noticed from this point. This is because the PMSM in the ECS acquiring a linearly increasing power from the generator as shown in Figure 10 and Figure 11. The linearly increasing AC currents resulted in correspondingly a linearly voltage drop in the transmission lines and at AC terminals of ATRUs. This results in a linear decrease of the DC voltage  $v_{HVDC1}$  and  $v_{HVDC2}$ . When the rated loads of ECS1 and ECS2 are applied, a slight voltage drop can also be seen in  $v_{HVDC1}$  and  $v_{HVDC2}$  at  $t=0.5s$ . The reduction of the WIPS power requirement at  $t=0.7s$  results in a higher  $v_{HVDC1}$  and  $v_{HVDC2}$ . When the generators SG1 and SG2 are connected,  $v_{HVDC1}$  and  $v_{HVDC2}$  decrease due to the difference between  $v_{HVAC1}$  and  $v_{HVAC2}$ . With SG1 removed from the system at  $t=1.02s$ , SG2 starts to supply the whole load system and the system comes to the steady state after a short transient period. The  $v_{HVDC1}$  and  $v_{HVDC2}$  from ABC and DP modelling techniques are well-matched as shown in Figure 9.

Since the system is assumed to be balanced, the currents flowing into ATRUs  $i_{ATRU1}$  and  $i_{ATRU2}$  are represented by the phase A current only. For comparison studies, the variables in the DP model are



transformed to the time domain. The simulation results of  $i_{ATRU1}$  and  $i_{ATRU2}$  are shown in Figure 10 and Figure 11. The currents  $i_{ATRU1}$  and  $i_{ATRU2}$  remains zero until the load connected to the HVDC buses at  $t=0.15s$ . The acceleration of PMSMs of ECS's induces the increasing of  $i_{ATRU1}$  and  $i_{ATRU2}$  starting from  $t=0.2s$ . The application of rated loads at ECS's introduces the step of  $i_{ATRU1}$  and  $i_{ATRU2}$ . It can be seen that the results from ABC and DP models are well matched during the whole simulation process. The magnitude of DPs  $\langle i_{ATRU1} \rangle_1$  and  $\langle i_{ATRU2} \rangle_1$  are also shown in Figure 10 and Figure 11. From these two figures, it can be seen that the magnitudes of  $\langle i_{ATRU1} \rangle_1$  and  $\langle i_{ATRU2} \rangle_1$  give the envelope of the result from the ABC model.

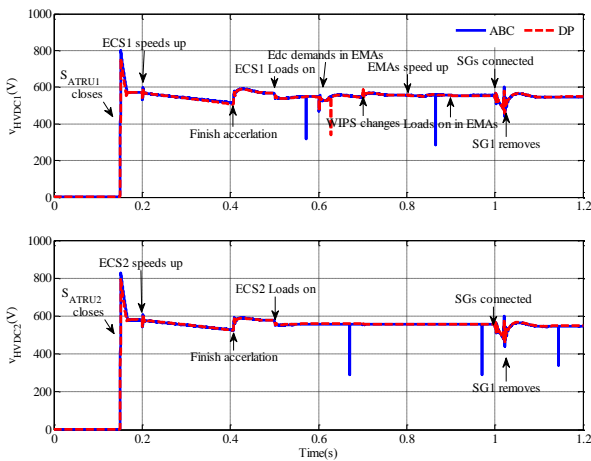


Figure 9 The dynamic response of  $v_{HVDC1}$  and  $v_{HVDC2}$ . Above: response of  $v_{HVDC1}$ ; below: response of  $v_{HVDC2}$

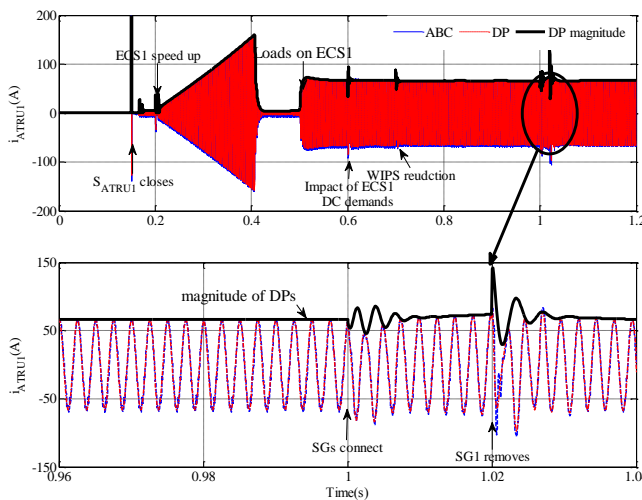


Figure 10 The dynamic response of  $i_{HVAC1}$ , phase A current flowing into ATRU1. Above:  $i_{HVAC1}$ ; below: zoom-in area of  $i_{HVAC1}$

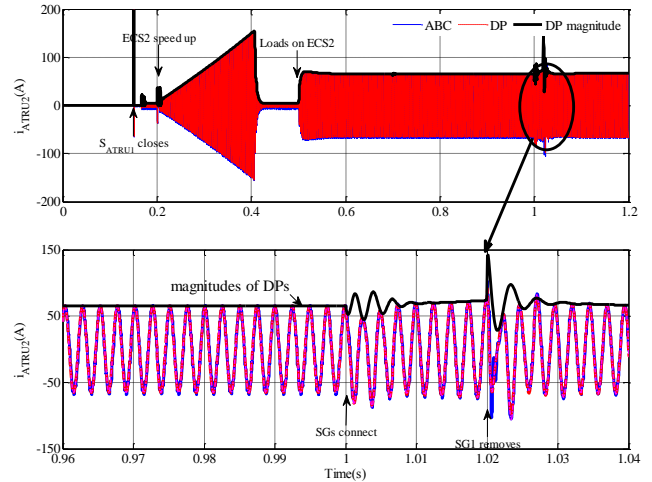


Figure 11 The dynamic response of  $i_{HVAC2}$ , phase A current flowing into ATRU2 Above:  $i_{HVAC2}$ ; below: zoom-in area of  $i_{HVAC2}$

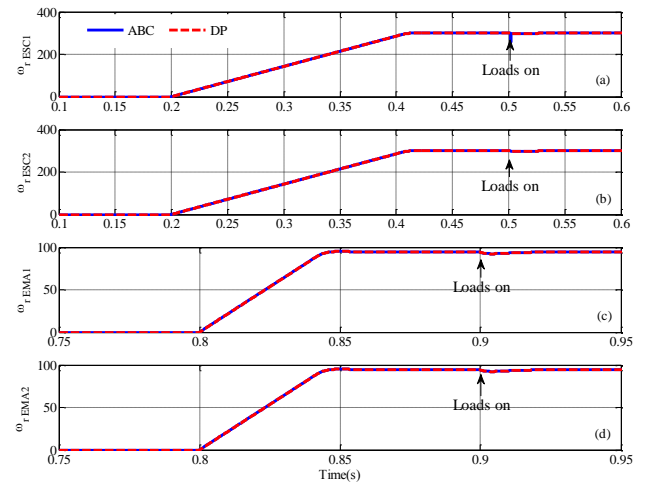


Figure 12 Dynamic response of drive loads, (a)  $\omega_{r\_ECS1}$  speed of PMSM of ECS1; (b)  $\omega_{r\_ECS2}$  speed of PMSM of ECS2; (c)  $\omega_{r\_EMA1}$  speed of PMSM of EMA1; (d)  $\omega_{r\_EMA2}$  speed of PMSM of EMA2

The speed of PMSMs of ECS1, ECS2, EMA1 and EMA2 is shown in Figure 12. The PMSMs are well controlled by their speed controllers. It presents well agreement between two modelling techniques.

Table 3 Comparison of the computation time between two different models

Model	ABC	DP
Simulation time (s)	7983	43
Acceleration	1	185

The computation time consumed by the two different models is compared in Table 3. It can be seen that the DP model is 185 times faster than the ABC model. The simulation time of ABC model is counted in hours compared with that of the DP model in seconds. Comparing the simulation time between the ABC and DP model, one can notice that the acceleration of DP model is significant. This is due to the fact that the variables are DC-like in the DP model which allows larger time steps in the simulation process. Another average modelling technique based using the  $dq$  frame (DQ0 model) can achieve the same acceleration order as that of DP model. However, the DQ0 model will lose its efficiency when the system is under unbalanced conditions. The DP model, on the other hand, can maintain its efficiency even the system is under unbalanced conditions. This merit of DP model has been demonstrated in our previous publications with subsystems of the aircraft EPS. Our future research will compare the three modelling techniques, ABC, DQ0 and DP models, with EPS system under MOET architecture.

## 5 Conclusion

The paper introduced a DP library for EPS studies. This library is developed in Dymola/Modelica. Based on the DP library established, a twin-generator EPS based on MOET architecture has been studied under normal conditions. The comparison between the ABC model and the DP model illustrated the accuracy of the developed DP model, during generator starting, parallel operation, load change and other events. The simulation time comparison revealed that the DP model is about 200 times faster than the ABC model. The accuracy and efficiency shows great potential of the DP modelling technique to be applied in EPS studies. In addition, since the DP modelling technique is essentially a frequency-domain modelling technique and its efficiency is not affected by the system condition, this allows the DP model a faster simulation speed even when the EPS is in unbalanced or faulty conditions. This merit of DP models will be shown in our future publications.

## References

1. Emadi, K. and M. Ehsani, *Aircraft power systems: technology, state of the art, and*

- future trends*. Aerospace and Electronic Systems Magazine, IEEE, 2000. **15**(1): p. 28-32.
2. Wu, T., *Integrative System Modelling of Aircraft Electrical Power Systems*. 2010, University of Nottingham.
3. B.Bachmann, H.W., *Advanced Modeling of Electromagnetic Transients in Power Systems*. Modelica Workshop 2000 Proceedings, 2000.
4. Wiesmann, H.
5. Sanders, S.R., et al., *Generalized averaging method for power conversion circuits*. IEEE Transactions on Power Electronics, 1991. **6**(2): p. 251-259.
6. Tao, Y., S. Bozhko, and G. Asher. *Assessment of dynamic phasors modelling technique for accelerated electric power system simulations*. in *Power Electronics and Applications (EPE 2011), Proceedings of the 2011-14th European Conference on*. 2011.
7. Tao, Y., S. Bozhko, and G. Asher. *Modeling of active front-end rectifiers using dynamic phasors*. in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*. 2012.
8. Tao, Y., S. Bozhko, and G. Asher. *Multi-generator system modelling based on dynamic phasor concept*. in *Power Electronics and Applications (EPE), 2013 15th European Conference on*. 2013.
9. Yang, T., S. Bozhko, and G. Asher. *Dynamic phasor modeling of autotransformer rectifier units for more-electric aircraft*. in *Power Electronics and Motion Control Conference (IPEMC), 2012 7th International*. 2012.
10. Yang, T., S. Bozhko, and G. Asher. *Modeling of active front-end rectifiers using dynamic phasors*. in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*. 2012.
11. Tao Yang, S.B., Greg Asher, *Modeling of An 18-pulse Autotransformer Rectifier Unit with Dynamic Phasors*. SAE 2012, 2012.
12. Jomier, T. *More Open Electrical Technologies Techincal Report*. 2009; Available from: <http://www.eurtd.com/moet/>.

# Flexible modeling of electrical power systems – the Modelica PowerSystems library

Rüdiger Franke, ABB, Germany – Ruediger.Franke@de.abb.com,  
Hansjürg Wiesmann, Switzerland – HJ.Wiesmann@bluewin.ch

## Abstract

New trends such as renewable power, virtual power plants, electric mobility and smart grids raise the importance of electrical power systems. The systems are manifold, including e.g. DC, single- and multi-phase AC with fixed and variable frequency. Often times such systems cover other physical domains as well, such as rotational mechanics and thermo-fluid. Required system models range from simple flow calculations of active power to detailed transient and asymmetric studies of three-phase systems. Transformed modal coordinates play an important role for the treatment of three-phase AC systems.

The paper introduces the new Modelica PowerSystems library. It covers arbitrary phase systems in one modeling framework. Besides simple generic models that are valid with all phase systems, also large sets of detailed component models for DC and three-phase AC are included. The detailed component models have been ported from the former Spot library to PowerSystems.

The usefulness of the library is shown on behalf of manifold examples and applications carried out so far for power/frequency control, intraday optimization and virtual power plants as well as electrical drive trains of wind turbines.

**Keywords:** *Modelica, electrical systems, power systems, dq0, quasi-static, transient, three-phase AC, DC, wind turbines, virtual power plants, electric mobility.*

## 1 Introduction

The Modelica Standard Library, version 3.2.1, contains the Electrical.Digital and Electrical.Analog sub-libraries, besides Electrical.Spice3 for digital and analog electronics. The Electrical.Multiphase and Electrical.Machines sub-libraries cover detailed

dynamic models of power electronics in the time domain [1]. The Electrical.QuasiStationary sub-library treats AC systems with complex phasors [2].

Some more electrical libraries have been developed during the last years by different authors. One of them, the Spot library, introduced a couple of important features, such as the uniform treatment of natural (abc) and modal (dq0) coordinates in one framework, making it well suited for the detailed modeling of transient effects [3]. Moreover the Spot library supported the per-unit system for parameterization and it contained an extensive set of component models.

Being well suited for detailed transient modeling of three-phase power systems, the Spot library has sometimes been considered overkill if simpler quasi-static models were sufficient. A further drawback of the Spot library was that the same models appeared multiple times in distinct sub-libraries for particular phase systems. The Spot library hasn't been available since Modelica version 3 anymore.

This paper introduces the new PowerSystems library that integrates the Spot library with a new concept of replaceable phase systems. The aim is to support different single and poly phase systems and different mathematical formulations in one framework. In particular this shall cover systems like:

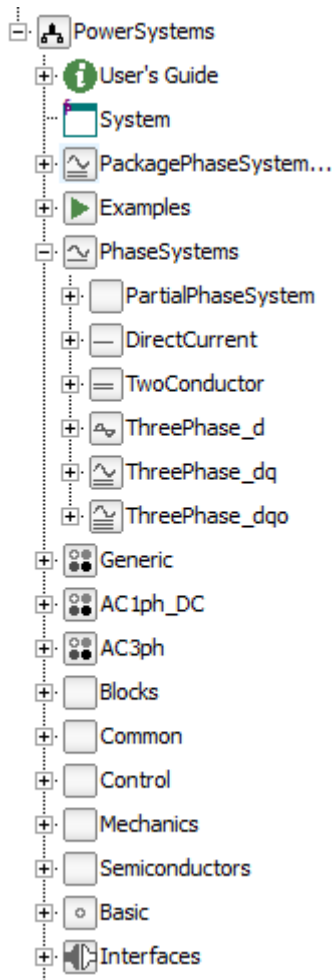
- AC systems, including steady-state, transient, and asymmetric,
- Variable frequency systems, e.g. in wind turbines or for drive control, and
- DC power systems, like HVDC

The replaceable phase systems also enable the application of one and the same library with one universal connector to different levels of detail. This starts from simple power balances and ranges through quasi-static models up to the treatment of detailed transient effects.

## 2 Library Structure and Interfaces

### 2.1 Library Structure

Figure 1 shows the package structure of the PowerSystems library.



**Figure 1: Library structure of PowerSystems**

The main packages are:

- **Examples** cover textbook power flow calculations, a demo for power/frequency control and introductory tutorials to the detailed component models;
- **PhaseSystems** defines different mathematical representations of electrical systems;
- **Generic** contains simple component models that can be used with any phase system for basic investigations;
- **AC1ph\_DC** contains detailed component models for DC and one phase AC;
- **AC3ph** contains detailed component models for three phase AC;

- **Blocks** contains signal oriented models, such as Multiplex and Transforms;
- **Control** contains special control blocks, such as Exciters or Governors;
- **Mechanics** provides TurboGroups, complementing a generator with rotor and gears;
- **Semiconductors** define required components such as Diodes and Thyristors.

### 2.2 Interfaces and PhaseSystems

The interfaces define a general power terminal.

```
connector Terminal
  replaceable package PhaseSystem;

  PhaseSystem.Voltage
    v[PhaseSystem.n];

  flow PhaseSystem.Current
    i[PhaseSystem.n];

  PhaseSystem.ReferenceAngle
    theta[PhaseSystem.m]
    if PhaseSystem.m > 0;
end Terminal;
```

The connector contains a replaceable PhaseSystem package making it applicable to different mathematical representations. The PhaseSystem defines:

- Number  $n$  of independent current and voltage components
- Number  $m$  of reference angles
- Types used in the connector (Voltage, Current, ReferenceAngle) so that terminals of different phase systems cannot be directly connected
- Functions
  - $j$  “operator”
  - angle and phase
  - phase quantities for voltage, current and power
  - system quantities for voltage, current and power

The vector of reference angles  $\theta[m]$  allows the definition of a rotating reference system for the description of AC systems with modal components. It is known from the Spot library that this enables the treatment of modal quantities in the time domain, covering transient and asymmetric systems as well.

The power Terminal is overdetermined with the reference angles though. This is treated with the operators `Connections.root`, `Connections.potentialRoot`, `Connections.isRoot` and `Connections.branch`. A Modelica tool needs to analyze connection graphs and eliminate redundant equations.

The following table summarizes the PhaseSystems that are predefined in the PowerSystems library:

PhaseSystem	n	m	Description
DirectCurrent	1	0	One voltage and one current component in natural coordinates
TwoConductor	2	0	Two voltage and two current components for Spot AC1ph_DC components
ThreePhase_d	1	0	One modal component for active power — like DirectCurrent, but converting voltage values to three phase
ThreePhase_dq	2	1	Two modal components for active and reactive power; one reference angle for frequency — cf. complex phasors with variable frequency
ThreePhase_dqo	3	2	Three modal components for active, reactive and dc power; two reference angles for Spot dqo components

A generic steady-state impedance model can be defined as:

```

model GenericInductiveImpedance
  replaceable package PhaseSystem =
    PackagePhaseSystem;
  function j = PhaseSystem.j;
  Terminal terminal_p(
    redeclare package PhaseSystem =
      PhaseSystem);
  Terminal terminal_n(
    redeclare package PhaseSystem =
      PhaseSystem);
  PhaseSystem.Voltage v[:] =
    terminal_p.v - terminal_n.v;
  PhaseSystem.Current i[:] =
    terminal_p.i;
  PhaseSystem.Frequency w =
    
```

```

    der(PhaseSystem.angle(
      terminal_p.theta));
  parameter
    Modelica.SIunits.Resistance
    R = 1 "active component";
  parameter
    Modelica.SIunits.Inductance
    L = 1/50
    "reactive component";
  equation
    v = R*i + w*L*j(i);
    zeros(PhaseSystem.n) =
      terminal_p.i + terminal_n.i;
    if PhaseSystem.m > 0 then
      terminal_p.theta =
        terminal_n.theta;
      Connections.branch(
        terminal_p.theta,
        terminal_n.theta);
    end if;
  end GenericInductiveImpedance;
  
```

Note the use of the function `j` that generalizes complex calculations known from quasi-static AC models to arbitrary phase systems. `ThreePhase_dq` with two modal components for active and reactive power defines a multiplication with the complex `j`:

```

function j
  input Real x[n];
  output Real y[n];
  algorithm
    y := {-x[2], x[1]};
  end j;
  
```

The simpler `ThreePhase_d` neglecting reactive power defines:

```

function j
  input Real x[n];
  output Real y[n];
  algorithm
    y := zeros(n);
  end j;
  
```

The more detailed `PhaseSystem_dqo` also considers a component for dc power in asymmetric systems, besides active and reactive power. It defines:

```

function j
  input Real x[n];
  output Real y[n];
  algorithm
    y := cat(1, {-x[2], x[1]},
      zeros(size(x,1)-2));
  end j;
  
```

The GenericInductiveImpedance model adapts to the selected PhaseSystem.

### 3 Component models and Examples

The PowerSystems library contains simple examples, including textbook power flow calculations, demonstration of power/frequency control, and tutorials for the modeling of power system transients.

#### 3.1 Network flow calculations

PowerSystems.Examples.Network contains examples that treat power flow with the quasi-static ThreePhase\_dq by using simple generic component models.

Figure 2 shows the Generic component models used by these examples.

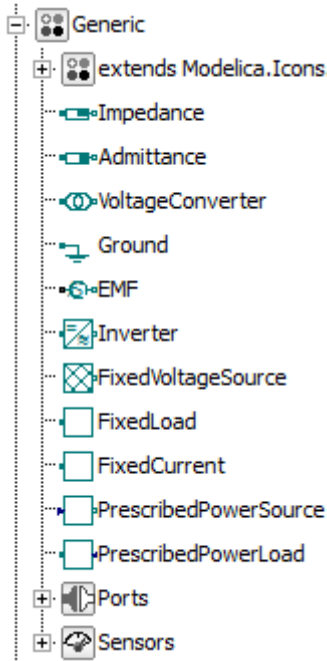


Figure 2: Generic component models

The NetworkLoop example implements a simple power flow model; see Figure 3. The textbook example is taken from [4].

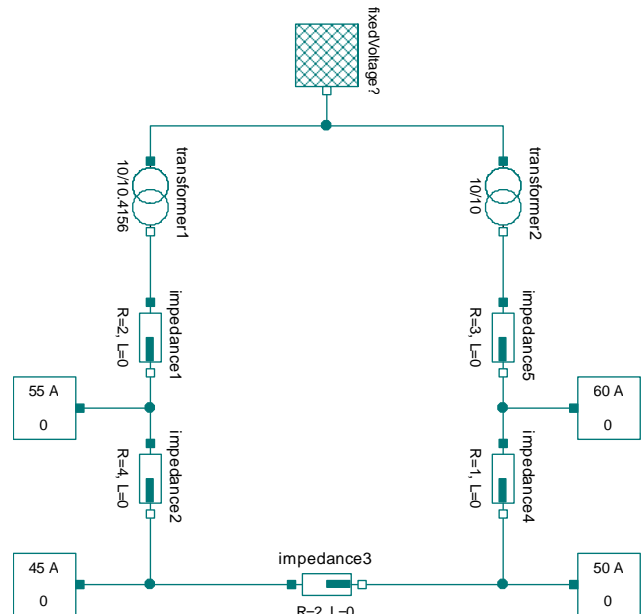


Figure 3: NetworkLoop example

The example demonstrates that the PowerSystems library enables the direct implementation of power flow models using established graphical representations. The graphical representation differs from electrical circuits as treated e.g. with the Electrical.Analog library. The FixedVoltageSource has only one terminal, as opposed to two pins in the Electrical.Analog library. Moreover there is no Ground component needed to treat loops.

#### 3.2 PowerWorld

PowerSystems.Examples.PowerWorld demonstrates the principles of power/frequency control; see Figure 4.

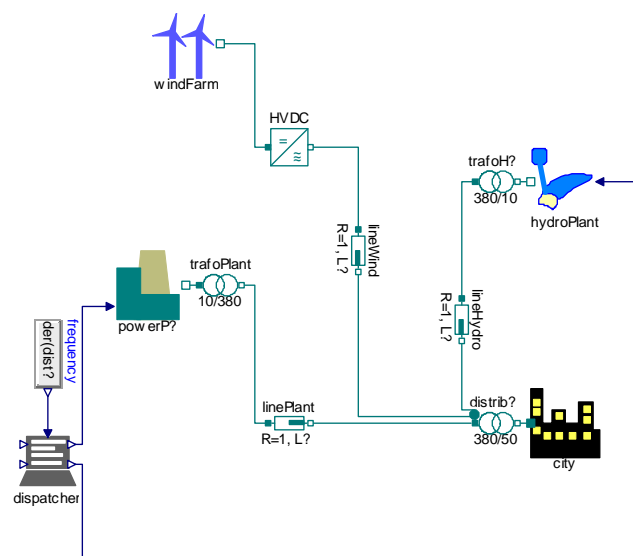


Figure 4: PowerWorld example

The example uses Generic components with the quasi-static ThreePhase\_dq because fast electrical transients and asymmetries are neglected. Rotating masses in the power plants and active power/frequency control determine the system dynamics.

Figure 5 shows simulation results covering one day of operation.

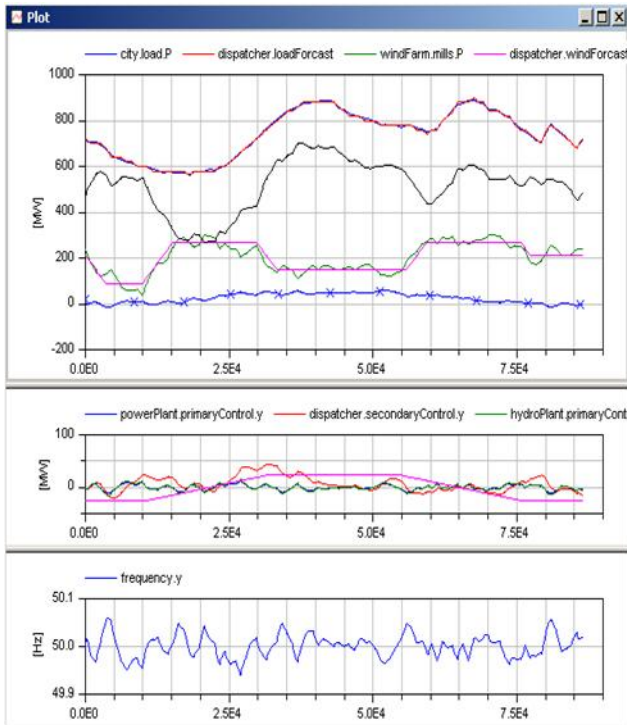


Figure 5: Simulation results for the PowerWorld example

The wind farm introduces disturbances into the system, caused by deviations of the actual weather from the forecast – see forecast in magenta vs. actual feed-in in green in the upper plot. A misbalance between production and consumption leads to deviations of the grid frequency from the nominal value of 50 Hz. The conventional power plant and the hydro plant accommodate for deviations with primary frequency control by means of proportional control. The remaining control error is exploited by the load dispatcher to determine the misbalance and to send out set points for secondary frequency control. This way the fast primary frequency control is relieved with the slower secondary frequency control, so that subsequent disturbances can be treated as well. The overall balance between production and consumption is maintained.

The example does not cover tertiary frequency control and intraday optimization that form further control loops on top of secondary and primary frequency control – see also the application in section 4.2 below.

### 3.3 Spot examples

PowerSystems.Examples.Spot has been ported from the Spot library [3]. They use the components from AC1ph\_DC and AC3ph; see Figure 6.

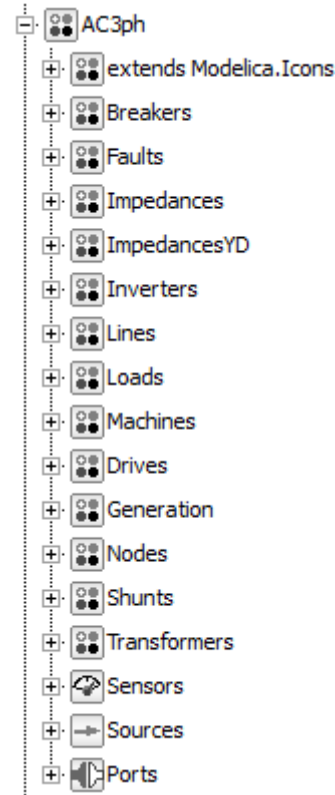


Figure 6: AC3ph component models

The extensive AC3ph components range from simple impedances through special devices such as breakers up to transformers, inverters and electrical machines. Moreover they cover composed models for Drives and Generation.

The Spot examples serve as tutorials for detailed transient modeling of electrical power systems. Figure 7 shows the fault clearance by short-time line switched off as example.

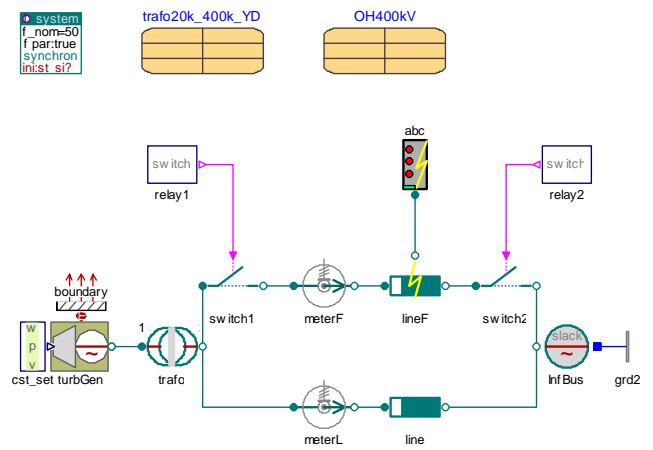


Figure 7: DoubleRXline example

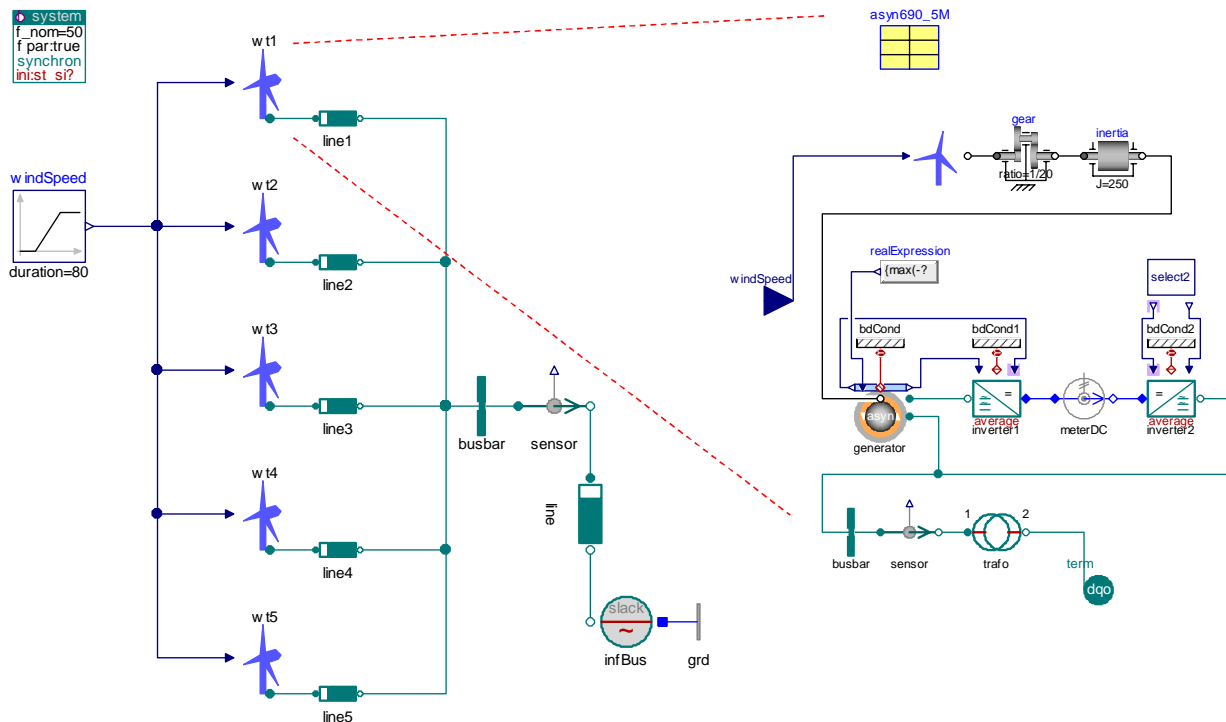


Figure 8: Wind farm model (electrical side)

## 4 Applications

### 4.1 Drive trains of wind turbines

The ITEA research project MODRIO aims for model driven online application. One application area addresses wind turbines. They are challenging because multiple physical domains need to be covered to master the overall system dynamics. The PowerSystems library has been used to implement transient models of electrical drive trains. Figure 8 shows an example with doubly fed induction generator (DFIG) on the right hand side. Multiple such wind turbine models have been assembled to a wind farm model as shown on the left hand side.

The electrical model will be integrated with a model for the mechanical side by SIMPACK.

### 4.2 Intraday optimization of municipal power

Traditional power/frequency control runs into limitations when facing large uncertainties due to increasing use of renewable energy. Intraday optimization reacts on new conditions by re-planning the power production. Such an intraday optimization was developed within the project econnect Germany for the investigation of the integration of emerging electric mobility with power generation.

The PowerSystems library has been used as basis for an optimization library. The optimization library predefines component models that contain, besides physical equations, also optimization constraints and objective terms. This way optimization programs can be implemented graphically like simulation models.

Figure 9 shows an exemplary model. The library is shown as tree on the left hand side. The intraday optimization treats a virtual power plant (VPP) consisting of multiple power generation units, such as wind, solar, hydro, as well as combined heat and power plants. Available storage capacities include heat buffers and a large battery. Moreover the exploitation of electric cars and of heat pumps as controllable consumers is investigated.

The Modelica model is translated to C code and exported as Functional Model Unit (FMU) to the ABB control system. The FMU is used a basis for a large-scale mixed integer optimization program, providing a model predictive control of the VPP. The online optimization approach is further discussed in [5].

Figure 10 shows the result of one optimization run. Dotted lines mark the original day-ahead plan. Caused by a surplus of wind production, the intraday optimization reduces the use of combined heat and power BHKW plants (green areas) until the heat buffers reach their lower limit (red areas). A battery that is installed in a parking house is charged at times when there is too much power in the grid. More details of the intraday optimization are given in [6].



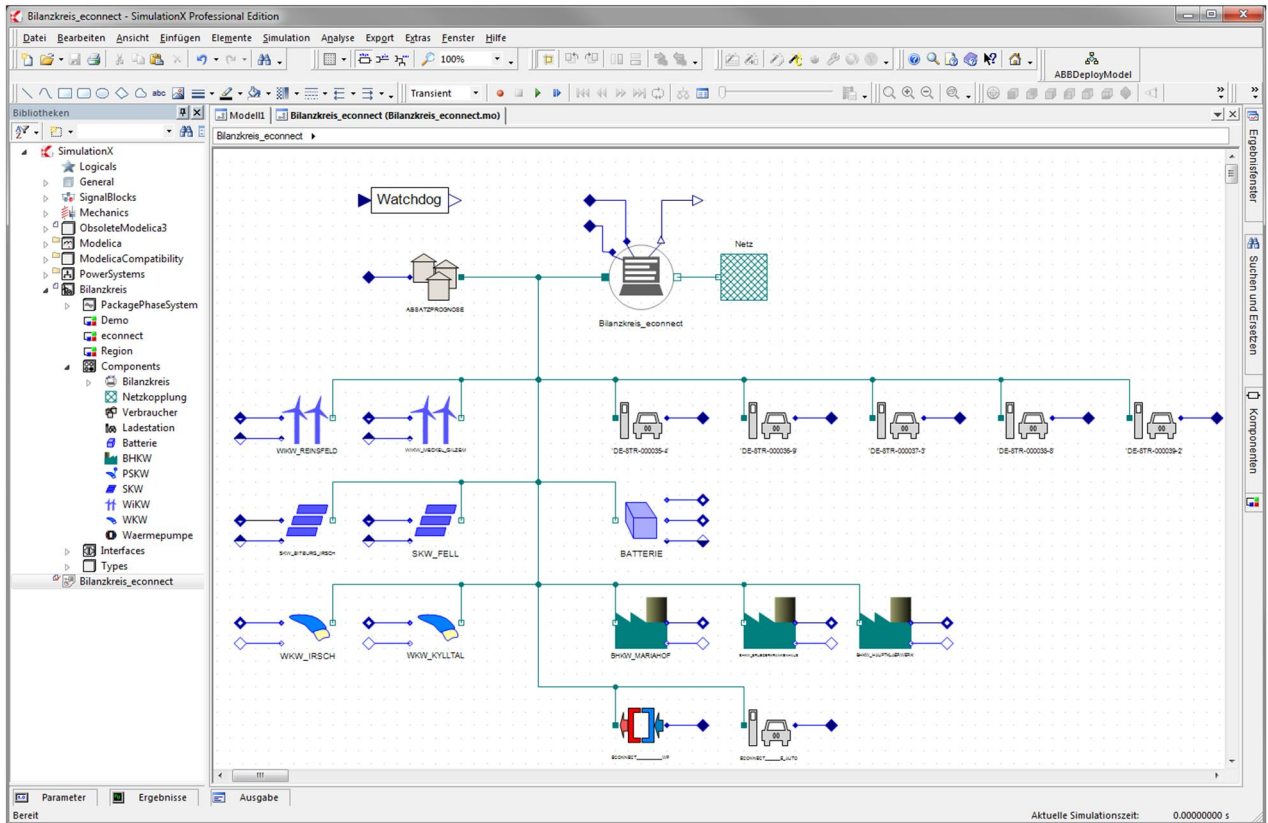


Figure 9: Optimization model for municipal power

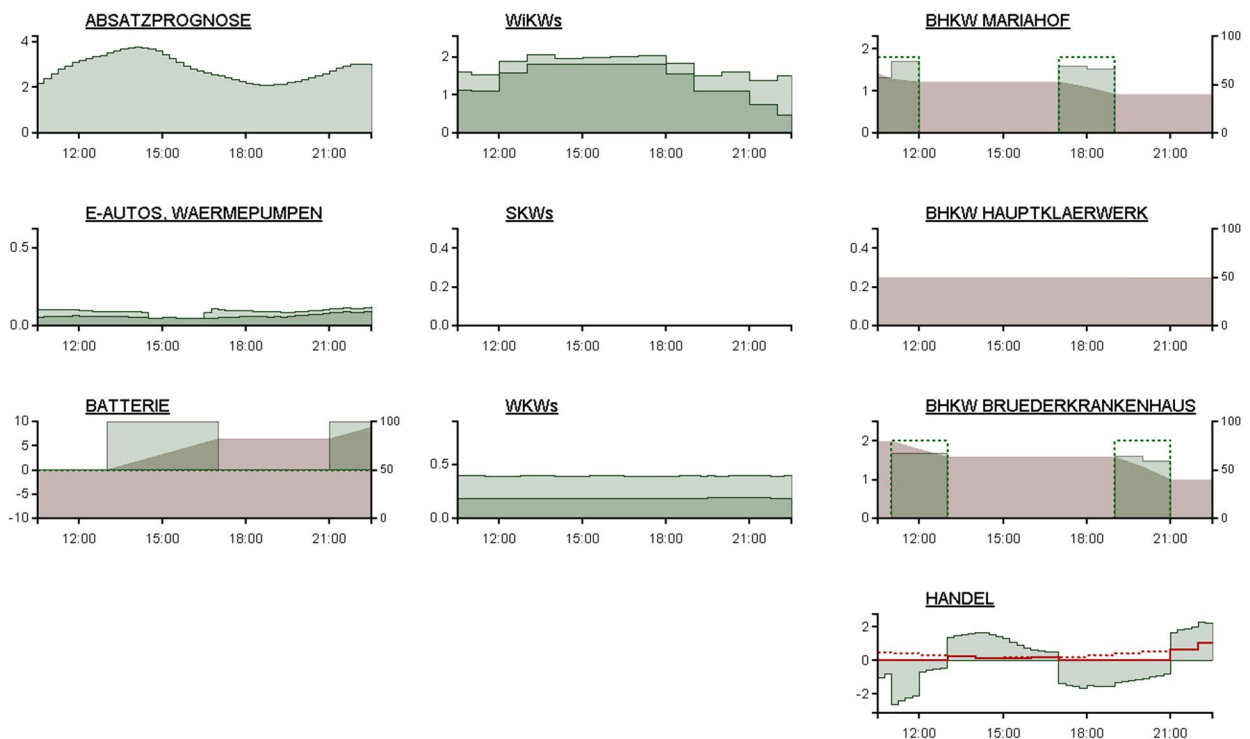


Figure 10: Exemplary intraday optimization result

## 5 Conclusions

The PowerSystems library started as a concept study for defining configurable phase systems, together with simple generic component models and universal power connectors. Later on the Spot library was integrated, so that extensive sets of detailed component models for DC and three-phase AC are available.

This way PowerSystems provides the comprehensiveness of Spot on the one hand side. Moreover, the new replaceable phase systems enable models at different level of detail, including also simple models for active power balances, besides symmetric and asymmetric AC models for active and reactive power, both steady-state and transient. A couple of examples and applications show the usefulness of the library at different levels of detail.

The PowerSystems Library is available under the Modelica license, version 2, at: <https://github.com/Modelica/PowerSystems>

Given sufficient interest by others, future work shall address the integration of the PowerSystems library or its concepts with the Modelica Standard Library.

## Acknowledgements

We thank Martin Otter who has converted the original Spot library to Modelica 3. This significantly simplified its integration with PowerSystems.

This work was supported in parts by BMBF within the ITEA2 project MODRIO (Model Driven Physical Systems Operation) – project id: ITEA 2 – 11004.

The project econnect Germany is funded by the federal ministry for economy and technology due to a decision of the federal parliament.

## References

- [1] C. Kral, A. Haumer: Modelica libraries for dc machines, three phase and polyphase machines, *Modelica Conference 2005*, Hamburg, March 2005. [https://www.modelica.org/events/Conference2005/online\\_proceedings/Session7/Session7a2.pdf](https://www.modelica.org/events/Conference2005/online_proceedings/Session7/Session7a2.pdf)
- [2] A. Haumer, C. Kral, J.V. Gragg, H. Kapeller: Quasi-Static Modeling and Simulation of Electrical Circuits using Complex Phasors, *Modelica Conference 2008*, Bielefeld, March 2008. <https://www.modelica.org/events/modelica2008/Proceedings/sessions/session2d3.pdf>
- [3] B. Bachmann, H. Wiesmann: Advanced Modeling of Electromagnetic Transients in Power Systems, *Modelica Workshop 2000*, Lund, September 2000. <https://modelica.org/events/workshop2000/proceedings/Bachmann.pdf>
- [4] D. Oeding, B.R. Oswald: *Elektrische Kraftwerke und Netze*, Springer, 7th edition 2011.
- [5] R. Franke, J. Doppelhamer: Integration of Advanced Model Based Control with Industrial IT. In: *Assessment and Future Directions of Nonlinear Model Predictive Control*. R. Findeisen, F. Allgöwer, L. Biegler (editors). Springer Verlag 2007.
- [6] R. Franke, S. Kautsch, M. Blaumann, L. Vogelbacher: Integration von erneuerbaren und konventionellen Energien – Intraday-Optimierung, Pooling und vorausschauende Zielsollwertführung, BWK 9/2013.

# Implementation of a Multi-Level Power Electronic Inverter Library in Modelica

C.I. Hill      P. Giangrande      C. Gerada      S.V. Bozhko

University of Nottingham

Aerospace Technology Centre, Innovation Park, Nottingham. NG7 2TU. UK.

[c.hill@nottingham.ac.uk](mailto:c.hill@nottingham.ac.uk)

[p.giangrande@nottingham.ac.uk](mailto:p.giangrande@nottingham.ac.uk)

## Abstract

This paper presents a newly developed Power Electronic Inverter library in Modelica. The library utilises a multi-level approach with increasing model complexity at progressively higher levels. All levels are fully interchangeable so as to provide a flexible library able to be utilised for investigation at single or multiple levels of complexity. Within this interchangeable multi-level approach, there are two key attributes which are implemented into this new library. The first is the ability to include losses between the input and output of the Power Electronic Inverter. This is implemented so that the losses are included irrespective of the direction of power flow. Secondly, this library also provides the ability to trigger single or multiple open and short circuit faults within the Inverter. The library therefore provides an extremely useful tool able to compare system response under a variety of operational scenarios.

*Keywords: Inverter; Multi-Level; Losses; Faults; Actuator*

## 1 Introduction

Power Electronic Inverters convert DC electrical power to AC. They are a vital part of the modern world and are becoming increasingly common due to the electrification of modern transport systems. They are used to drive the electrical machines within electric vehicles [1] and are an essential part of electrically driven actuator systems within the More Electric Aircraft [2]. Indeed the adoption of electrically driven actuators is becoming widespread among aircraft manufacturers due to the evident benefits in terms of efficiency, weight and maintenance [3]. However, despite the high efficiency of these electrical systems and low probability of fault occurrence, it is still essential to consider the losses that do occur and the effect of fault conditions. The `Inverters`

library presented here provides the ability to include losses, analyse thermal response and introduce fault conditions into the modelling environment. It also gives the user multiple interchangeable models of the Power Electronic Inverter in order to allow analysis at multiple levels of complexity. This is a new approach as the MSL, SPOT, Smart Electric Drives and Modelon's ElectricPower libraries do not provide this flexible multi-level approach with loss and fault capability [4][5]. The `Inverters` library presented here forms part of an overall `Actuator` library developed as part of Actuation 2015 [6]. For an overview of the entire `Actuator` library please see [7]. However the purpose of this paper is to give in-depth detail on the multi-level modelling of the Power Electronic Inverter.

At first the library structure will be presented along with a definition of the multiple modelling levels. Details of the individual models will be given including both commonalities and differences. Simulation results will be shown in order to demonstrate results which can be obtained using the 5 modelling levels.

## 2 Multi-level definition

Before detailing the actual models a definition of each modelling level must be given. This is defined in line with the overall definition established by all partners within Actuation 2015 [6]. Table 1 shows the agreed definitions which are used throughout the `Actuator` library [7] and are also used within this `Inverters` library.

Level	Colour
1	Perfect
2	Linear Effects, Invertible
3	Non-Linear Effects, Invertible
4	Hard Non-Linear Effects
5	Fully Switched Models

Table 1: Modelling level definition.

In addition common colour coding was agreed for each modelling level as detailed in Table 2.

Level	Colour
1	Red
2	Blue
3	Grey
4	Green
5	Dark Yellow

Table 2: Colour coding of each modelling level.

### 3 Structure

Figure 1 shows the structure of the `Inverters` library.

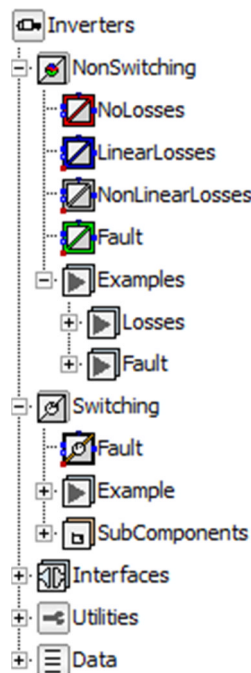


Figure 1: Structure of the `Inverters` package.

The `Inverters` package provides 5 levels of complexity in line with the multi-level definitions detailed in Section 2. As can be seen from Figure 1 the package is split into the `NonSwitching` package and the `Switching` package. The `NonSwitching` package provides colour coded models for levels 1 to 4 and the `Switching` package provides the level 5 model. Each level has a corresponding example of its use. The colour and level assignment of each model is in line with the definition of the modelling levels for the overall `Actuator` library as detailed previously in Section 2. The colours for each level and the icons for `NonSwitching` and `Switching` models are extended from the `Inverters.Utilities.Icons` package.

## 4 Model Interfaces

A central feature of this `Inverters` library is that each modelling level is fully replaceable with one another. In order for each modelling level to be fully interchangeable a common interface is used for all 5 modelling levels.

The electrical interfaces adopted for the DC interface are the Modelica Standard Library (MSL) `PositivePin` and `NegativePin`. On the AC side a `PositivePlug`, is used and it is assumed that a 3 phase AC side is used. A `RealInput[3]` is used for input of the 3 phase AC voltage demands. If closed loop control is used then the `RealInput[3]` input signals should be the unmodulated outputs of the current controller. A `PartialConditionalHeatPort` is used as a thermal interface for all levels. These interfaces are defined and then extended from the package `Actuator.Electrical.Interfaces`.

## 5 Non-Switching

### 5.1 Theoretical basis

There are two assumptions which are used as the basis of creating the `NonSwitching` package of Inverter models. The first assumption is that the AC output voltage signals, which in a `Switching` Inverter would be created via Pulse Width Modulation, are ideally created with a pure fundamental frequency exactly as demanded by the control signals. As such the unmodulated voltage demand signals can be used to ideally create the electrical Inverter output voltages. Within Modelica this simply means taking the 3 phase demand `RealInput[3]` and connecting it to a MSL `SignalVoltage` model, with  $m=3$ . This forms the initial basis of the Inverter model on the AC side and is constant throughout all the Inverter models within the `NonSwitching` package. Figure 2 shows this along with the inclusion of the second assumption detailed below.

The second assumption which forms a basis of the `NonSwitching` package is the power equivalence principle. This assumes that the power input to the Inverter is equal to the power output from the Inverter. As one side of the Inverter is AC and one is DC then:

$$P_{DC} = P_{AC} \quad (1)$$

In addition as:

$$P_{DC} = I_{DC} V_{DC} \quad (2)$$

Therefore substituting (2) into (1):

$$I_{DC} V_{DC} = P_{AC} \quad (3)$$

Where  $P_{DC}$  is the Inverter DC side power,  $P_{AC}$  is the Inverter AC side power,  $I_{DC}$  is the Inverter DC side current and  $V_{DC}$  is the Inverter DC side voltage.

Now, as both the AC power and the DC voltage can be measured, this means the DC current can be calculated by:

$$I_{DC} = \frac{P_{AC}}{V_{DC}} \quad (4)$$

The AC power is measured on the AC side using a MSL `PowerSensor` model with  $m=3$ . The DC voltage is measured using the MSL `VoltageSensor` model. However only the absolute value of the DC voltage is required and so a MSL `Abs` model is used. The calculated DC current is output using a MSL `SignalCurrent` model. This structure therefore creates the Level 1 Inverter and is the basis for the Level 2, 3 and 4 Inverters. It is summarised in block form in Figure 2. No thermal response is included within Level 1 as no losses exist.

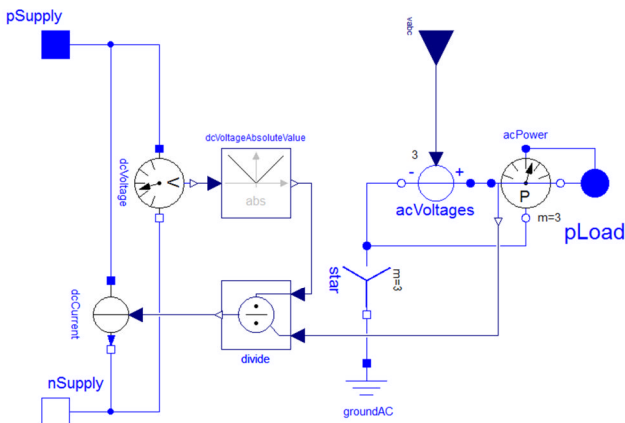


Figure 2: Basis of the Level 1 Inverter.

## 5.2 Losses

This sub-section now builds on the details given in Section 5.1 in order to create the level 2, 3 and 4 Inverter models.

To begin, it is important to note that all the Power Electronic Inverter models are fully multi-directional. As a result losses are implemented whether power flow is from the DC side to the AC

side or from the AC side to the DC side. Hence if power flow is from DC to AC the AC side power will be lower than the DC side and vice versa. This is made inherent within Levels 2, 3 and 4 by use of the MSL models shown in Figure 3. As can be seen from Figure 3 `Sign` is used to detect the sign of the AC power. This is then used, along with `Switch` and `GreaterEqualThreshold`, with `threshold=0`, in order to switch between two `Constant` sources. The `Constant` sources provide two differing values dependent on the efficiency of the Inverter and the direction of power flow as will be detailed below.

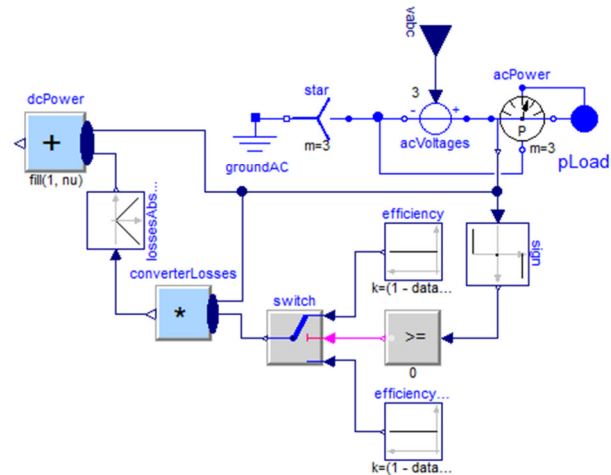


Figure 3: Basis of the Level 2 Inverter.

Irrespective of the direction of power flow, in order to calculate both the Inverter losses and the DC power the same power equivalence principle is used as detailed in Section 5.1. However, as Inverter efficiency is now included, (1) is now modified.

When power flow is from the DC side to the AC side (1) becomes:

$$\eta P_{DC} = P_{AC} \quad (5)$$

Where  $\eta$  is the efficiency of the Inverter.

When power flow is from the AC side to the DC side (1) becomes:

$$P_{DC} = \eta P_{AC} \quad (6)$$

However an important aspect of this model is that a thermal response is also given, as will be described in Section 5.3, and as such the power lost within the Inverter must be found. Therefore it is crucial to have both the power losses and the DC power in terms of the specified Inverter efficiency and measured AC power.

Efficiency is defined as:

$$\eta = \frac{P_{\text{OUTPUT}}}{P_{\text{INPUT}}} \quad (7)$$

And:

$$P_{\text{OUTPUT}} = P_{\text{INPUT}} - P_{\text{LOSSES}} \quad (8)$$

Where  $P_{\text{OUTPUT}}$  is output power,  $P_{\text{INPUT}}$  is input power and  $P_{\text{LOSSES}}$  is the power lost from the device between input and output.

Substituting (8) into (7) gives:

$$\eta = \frac{P_{\text{INPUT}} - P_{\text{LOSSES}}}{P_{\text{INPUT}}} \quad (9)$$

Making  $P_{\text{LOSSES}}$  the subject gives:

$$P_{\text{LOSSES}} = P_{\text{INPUT}}(1 - \eta) \quad (10)$$

This therefore gives power losses in terms of the input power. However using (7) and (8) the efficiency and hence power losses can also be defined in terms of the output power:

$$\eta = \frac{P_{\text{OUTPUT}}}{P_{\text{OUTPUT}} + P_{\text{LOSSES}}} \quad (11)$$

Hence in terms of  $P_{\text{LOSSES}}$ :

$$P_{\text{LOSSES}} = \frac{P_{\text{OUTPUT}}(1 - \eta)}{\eta} \quad (12)$$

Equations (10) and (12) are crucial as they define the power losses in terms of both input and output power. As a result, when power flow is from the DC to the AC side the AC side is the output and (12) gives:

$$P_{\text{LOSSES}} = \frac{(1 - \eta)}{\eta} P_{\text{AC}} \quad (13)$$

Whereas when power flow is from the AC side to the DC side then the AC side is the input and (10) gives:

$$P_{\text{LOSSES}} = (1 - \eta)P_{\text{AC}} \quad (14)$$

Within the Inverter models developed here, these two results are incredibly useful as in both cases the power losses are defined in terms of the measurable AC power and the specified efficiency. Hence, as shown in Figure 3 and described earlier, the AC power can be measured and multiplied by the coefficients shown in (13) and (14) in order to find the power lost. This is implemented using two `Constant` sources which are selected depending on the direction of power flow.

Next, the DC power must also be defined in terms of the specified efficiency and AC power. Therefore if power flow is from DC to AC then (8) becomes:

$$P_{\text{AC}} = P_{\text{DC}} - P_{\text{LOSSES}} \quad (15)$$

Substituting (13) for  $P_{\text{LOSSES}}$  and making  $P_{\text{DC}}$  the subject gives:

$$P_{\text{DC}} = P_{\text{AC}} + \frac{(1 - \eta)}{\eta} P_{\text{AC}} \quad (14)$$

Now if power flow is from AC to DC then (8) becomes:

$$P_{\text{DC}} = P_{\text{AC}} - P_{\text{LOSSES}} \quad (15)$$

Substituting (14) for  $P_{\text{LOSSES}}$  and making  $P_{\text{DC}}$  the subject gives:

$$P_{\text{DC}} = -P_{\text{AC}} + (1 - \eta)P_{\text{AC}} \quad (16)$$

If equations (14) and (16) are compared it can be seen that the only variation is the sign of the first  $P_{\text{AC}}$  term and the coefficient of the second  $P_{\text{AC}}$  term. Hence, as stated previously, both equations can be implemented within Modelica using two `Constant` sources which supply the coefficient terms and `Sign` to select between them.  $P_{\text{AC}}$  is naturally measured as positive or negative by `PowerSensor`.

Within Modelica (14) and (16) become:

$$dcPower = acPower + ((1 - data.eta)/data.eta) * acPower$$

And:

$$dcPower = acPower + (1 - data.Eta) * acPower$$

Finally, as the DC voltage can be measured, in both cases the DC current is then calculated from the DC power using (2) and implemented in Modelica as shown on the left hand side of Figure 2.

### 5.3 Efficiency

The previous sub-section showed how losses are included within the `NonSwitching Inverter` models. All losses were specified in terms of Inverter efficiency. However within the Level 2 model this efficiency is specified in a different manner to the Level 3 and 4 models.

When using the level 2 model the user is able to specify a constant efficiency for the Inverter. However if no value is specified by the user then a default value is used. This is then used under all operating conditions and is a constant throughout. The losses are not constant as they depend on the operating

conditions but the efficiency is constant. As detailed in Section 5.2 these are allowed for irrespective of the direction of power flow.

Levels 3 and 4 both use non-linear efficiency characteristics. The user is able to specify the power range of the Inverter and also the losses within the Inverter over the specified power range. A default characteristic is also given, as shown in Figure 4 below. However accurate characteristics are recommended to be added by the user if known. The default characteristics are per unit and are scaled within the model by the specified, or default, power range.

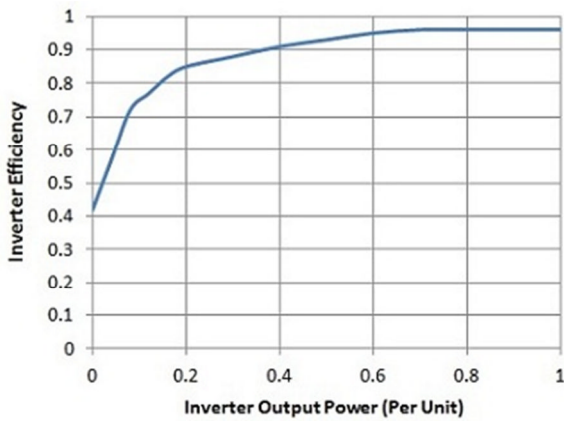


Figure 4: Default non-linear loss characteristics

These non-linear losses are implemented using CombiTable1D instead of Constant. Table 3 summarises the losses included in each level of the Inverters package. The Level 5 losses will be discussed in Section 6.

Level	Losses
1	None
2	Linear Losses
3	Non-Linear Losses
4	Non-Linear Losses
5	Conduction Losses

Table 3: Losses modelled within each level of the Inverters package.

An example system is now used to demonstrate the use of the Inverters package and show the variation in Inverter losses between levels. The Inverter is used within a system to control the speed of a Permanent Magnet Machine. The Inverter is fed from a MSL ConstantVoltage and supplies a MSL SM\_PermanentMagnet. An inner current loop is used with an outer speed control loop. Please note the controllers used are not part of the Inverters package. The overall system is shown in Figure 5.

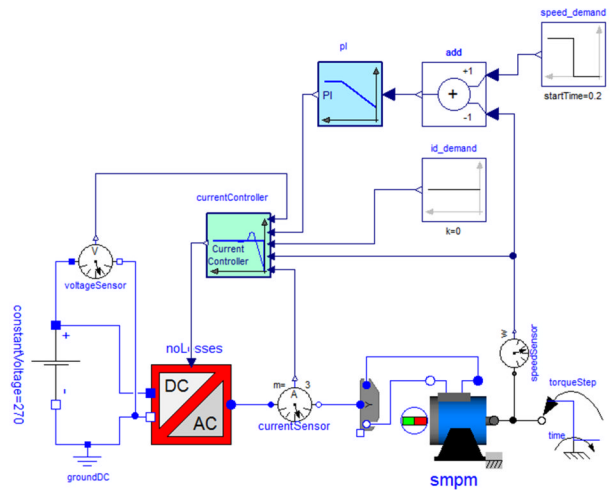


Figure 5: Example use of Inverters package.

Figure 6 shows the speed response of the system when an initial load torque is demanded at 0 seconds and a step in speed from 0 to 200 rad/s is demanded at 0.05s. A further increase in load torque is demanded at 0.14s. Figure 7 shows the 3 phase AC current response.

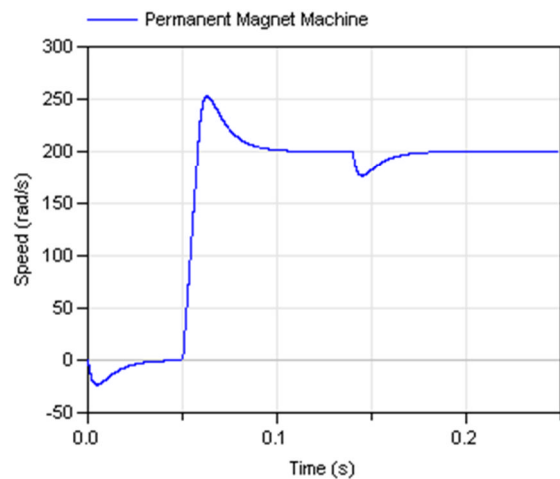


Figure 6: Speed response

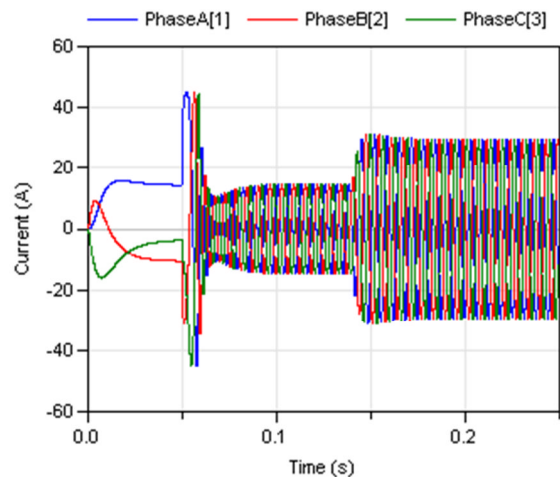


Figure 7: 3 phase current response

Using this response the Levels 1 to 4 Inverters are simulated and the losses generated over the simulation period are compared in Figure 8.

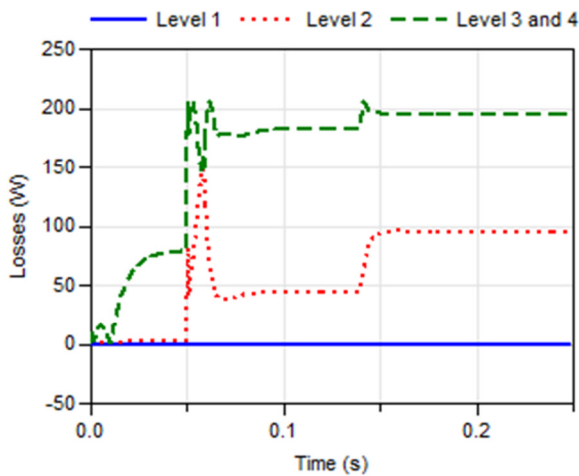


Figure 8: Losses produced in Levels 1, 2, 3 and 4.

It can be seen from Figure 8 that the losses within the Inverter models are substantially different depending on the level of the Inverters package used. It can be seen how, in this case, the non-linear losses in Levels 3 and 4 are always higher. This is due to the Inverter operating at low to medium power. It can also be seen that the increase in losses at 0.14s is greater for the Level 2 Inverter than the Level 3 and 4 Inverters. At 0.14s the increase in losses in the Level 2 Inverter is greater as the losses are proportional to the Inverter power. However within the Level 3 and 4 Inverters the increase is less due to the non-linear loss characteristics meaning the Inverter efficiency is higher at higher power. This therefore shows how the inclusion of non-linear losses could be important. However, if accurate non-linear information is not known, operation is known to be in a linear region, or losses are not required then Level 1 or 2 Inverters can easily be used.

### 5.4 Thermal Response

The Level 2, 3 and 4 Inverter models within the Inverters.NonSwitching package all have the same thermal implementation. As can be seen from Figure 9 a MSL PrescribedHeatFlow model is used to convert the calculated Inverter losses into heat flow. This is connected to a HeatCapacitor which represents the thermal capacitance of the Inverter switch modules. A ThermalConductor is used between the HeatCapacitor and the output PartialConditionalHeatPort in order to represent the thermal resistance between the switch module and its cooling system. A PartialCondition-

alHeatPort is used so losses can still be included if no thermal output is required.

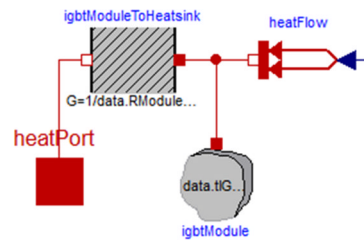


Figure 9: Thermal implementation within Levels 2, 3 and 4 Inverter models.

The user may specify the thermal parameters for the switch modules. All the required values can be read directly from module data sheets. If no user specified value is given then default values are used.

### 5.5 Fault

As detailed in the introduction the Inverters library presented here forms part of an overall Actuator library developed as part of Actuation 2015 [6]. A crucial aspect of the Actuator library is the inclusion of fault conditions within all the component models. In the case of the Power Electronic Inverters these are mainly included within the level 5 Switching model as no switches exist within the lower level models. However within the Level 4 Inverter a full bridge fault is possible. When triggered the Power Electronic Inverter output becomes zero. This represents a full-bridge open circuit fault or deactivation of the Inverter.

Figure 10 shows the speed response of the same system as shown in Figure 5. However in this case the Level 4 Inverter model is used and a full-bridge fault is triggered at 0.2s. Figure 11 shows the AC voltage response and Figure 12 shows the Inverter output power.

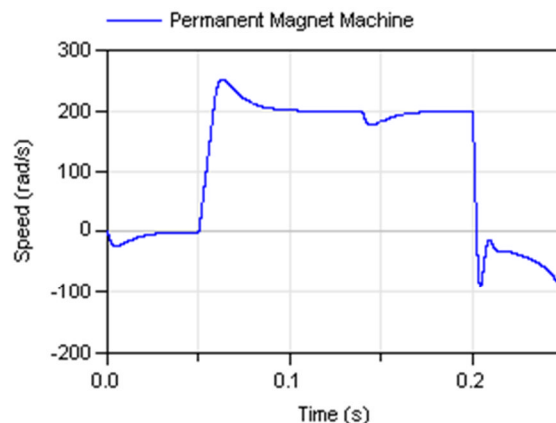


Figure 10: Faulted speed response



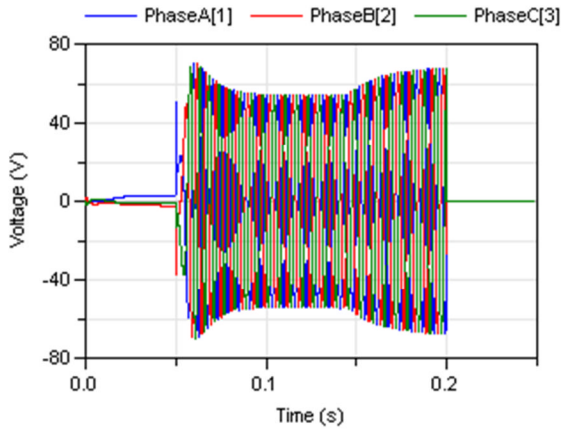


Figure 11: Faulted 3 phase voltage response

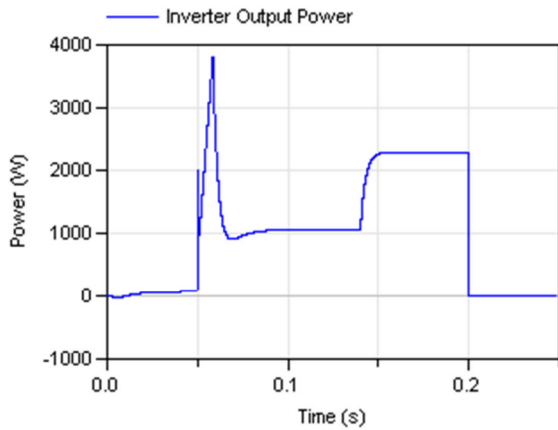


Figure 12: Faulted Inverter power response

It can be seen from Figures 10 to 12 that when the fault is triggered at 0.2s the Inverter output becomes 0. Therefore the output voltage and power both become 0 as shown in Figures 11 and 12. Within Figure 10 the speed response shows how, after the fault is triggered, the Permanent Magnet Machine accelerates in the negative direction due to the load torque acting as a mechanical source.

Table 4 summarises the fault conditions included in each level of the `Inverters` package. The Level 5 faults will be discussed and shown in Section 6.

Level	Faults
1	None
2	None
3	None
4	Full-Bridge
5	Single Switch, Single Phase, Multi-Phase and Full Bridge

Table 4: Fault conditions modelled within each level of the `Inverters` package.

The Failure Triggering Toolbox detailed in [7][8] is used within the models in order to trigger the required faults. A Boolean coding system is used to trigger the faults where 0 = normal operation, 1 = faulted operation. The faults can be implemented at any time instant but must be defined pre simulation.

## 6 Switching

### 6.1 Model Basis and Sub-Components

The Level 5 Inverter model within the `Inverters.Switching` package is based on the 6 switch, 3 leg topology shown in Figure 13. The switches are numbered S1 to S6 according to the order of switching. Each has a corresponding parallel diode.

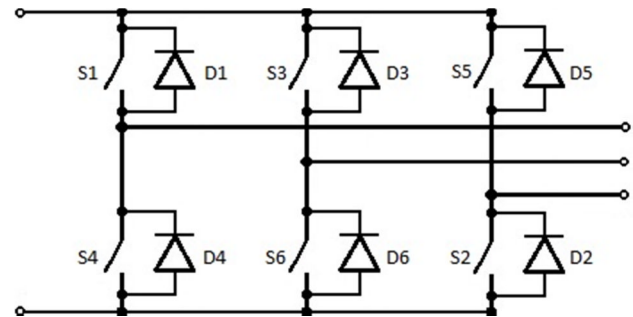


Figure 13: Topology used for `Inverters.Switching` package

In order to build the Level 5 Inverter certain sub-components were needed. These are contained in `Inverters.Switching.SubComponents` and include `FaultSwitch`, `PulseWidthModulation` and `FaultInjector`.

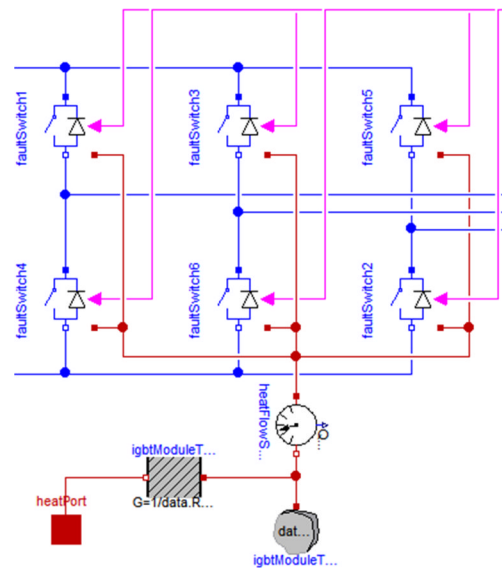


Figure 14: Basis of the Level 5 Inverter.

`FaultSwitch` is a model which represents the parallel switch and diode combination shown in Figure 13. Figure 14 shows how `FaultSwitch` is used within the Level 5 Inverter in order to create the topology shown in Figure 13.

Figure 15 shows the MSL models used within `FaultSwitch`. An `IdealClosingSwitch` MSL model is used as the main switch for power conversion. It receives a Boolean signal which dictates whether the switch is open or closed. In series with this switch is a `VariableResistor`. Under normal conditions the resistance of this `VariableResistor` is set to be a magnitude of  $10^{-6}$  smaller than the `IdealClosingSwitch` resistance when conducting. It therefore has negligible effect. However this `VariableResistor` is mainly included for use under fault conditions. As the `IdealClosingSwitch` resistance is very small, under short circuit conditions a very large current is obtained. In order to counteract this, the user is able to limit the short circuit current by defining a larger short circuit resistance. The `VariableResistor` is therefore used to introduce this larger resistance under fault conditions. A `BooleanToReal` MSL model is used as an input to the `VariableResistor`. This controls the resistance value as it receives a Boolean control signal which is false under normal operation and true under faulted operation. This does mean that the resistance also increases during an open circuit fault; however this has negligible effect due to the open circuit. The Boolean control signal comes from another `Inverters.Switching` sub-component named `FaultInjector` which will be detailed later in this section.

In parallel with the `IdealClosingSwitch` is an `IdealDiode`. The diode is uncontrolled and has default parameters, although these can also be specified by the user. This diode is also in series with the `VariableResistor` detailed above. As a result the increased resistance during short circuits, as detailed above, also affects the current path through the diode. During open circuit faults the combination of the diode and increased resistance provides a path for the release of otherwise trapped energy. If both paths were instantaneously open circuited then an extremely large voltage spike would be produced due to trapped energy. In a practical system this would result in sparks as the air would conduct under high voltage. However, in order to avoid these numerical spikes in this model the diode is allowed to briefly conduct.

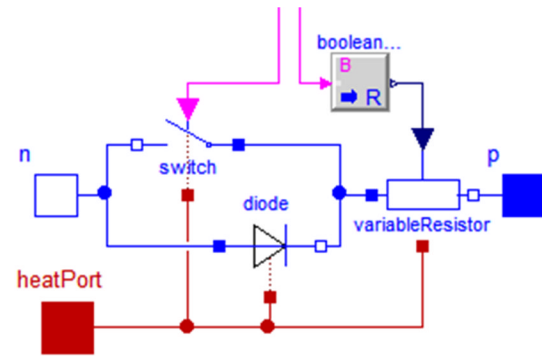


Figure 15: Inverters. Switching. Sub-Components. `FaultSwitch`

The `PulseWidthModulation` sub-component implements sine-triangle Pulse Width Modulation in order to create the 6 required switching signals. The user is able to specify the switching frequency and also the modulation index. In order to make all 5 modelling levels interchangeable the voltage demand signals which are input to the Level 5 Inverter are required to be un-modulated as was the case for Levels 1 to 4.

Finally, the `FaultInjector` sub-component is simply used to override the Pulse Width Modulated switching signals when a fault is triggered by the user. Under normal operation the switching signals are fed through the `FaultInjector` unchanged however when a fault is triggered the Pulse Width Modulated switching signals are blocked and the relevant fault state is output to the relevant switches. In addition the `FaultInjector` sub-component also generates the Boolean control signal which dictates the resistance of the `VariableResistor` contained within `FaultSwitch`. Boolean logic is used to create the relevant signal required to increase the resistance of the `VariableResistor` during faulted operation.

The Failure Triggering Toolbox detailed in [7][8] is used within `FaultInjector` in order to trigger the required faults. An Integer coding system is used to trigger the faults where 0 = normal operation, 1 = short circuit and 2 = open circuit. The faults can be implemented at any time instant but must be defined pre simulation.

## 6.2 Faults

Figures 16 to 21 show the effect of introducing faults within the level 5 Inverter. The same example system, speed demand and load torque is used as shown in Figure 5 and described in Section 5.3 except for the use of the Level 5 Inverter.

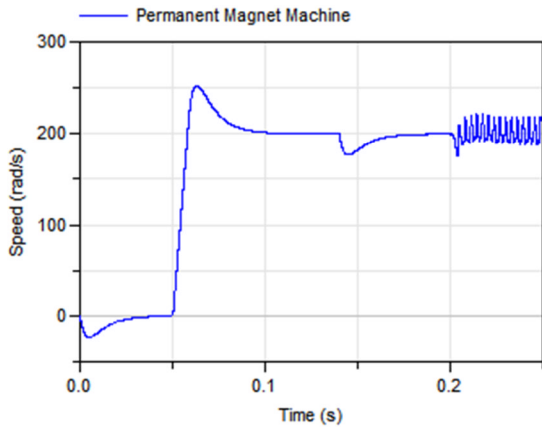


Figure 16: Speed response with a single phase open circuit fault at 0.2s

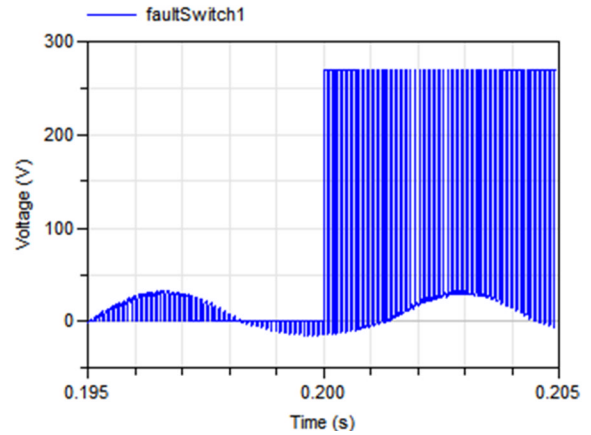


Figure 19: Current through FaultSwitch1 with a short circuit fault at 0.2s

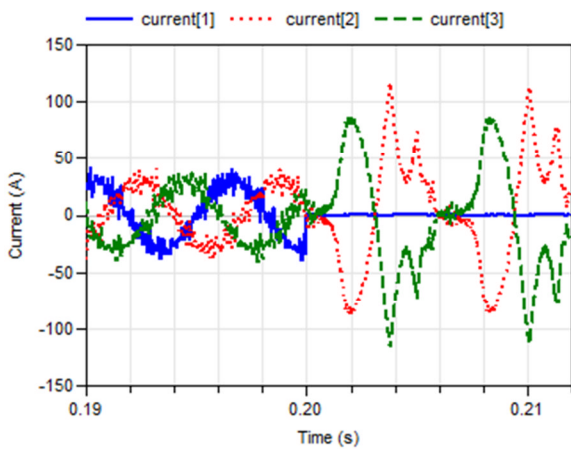


Figure 17: Current response with a single phase open circuit fault at 0.2s

Figures 16 and 17 show the effect of introducing a single phase fault. It can be seen that after the fault occurs the speed demand is still roughly maintained but with a lot of ripple and the faulted phase current falls to 0. The small current on the faulted phase is due to the dissipation of trapped energy as explained in Section 6.1.

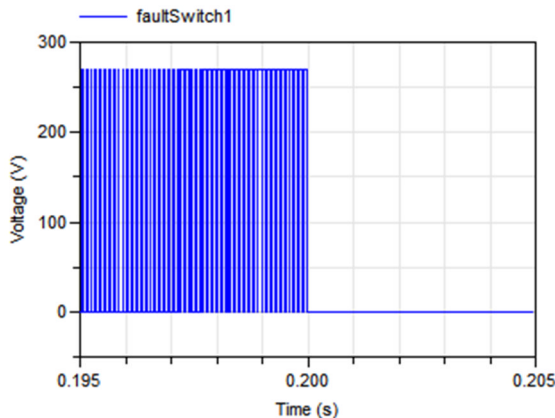


Figure 18: Voltage over FaultSwitch1 with a short circuit fault at 0.2s

Figures 18 and 19 show the effect of a single switch short circuit fault. Figure 18 shows the voltage over the switch. As expected, before the fault the voltage follows the Pulse Width Modulation signals but after the fault it is shorted to 0. Figure 19 shows the current through the same switch. As detailed in Section 6.1, this is limited by the specified short circuit resistance. In this case the specified short circuit resistance is  $1\Omega$  and as  $V_{DC}$  is 270V then the short circuit current is 270A. It can be seen that during the cycles of the Pulse Width Modulation where the switch should be open it continues to operate correctly.

Figure 20 shows the current response for a full bridge open circuit. As expected all 3 phase currents fall to 0 when the fault is triggered at 0.2s. Finally, Figure 21 shows the speed response for a full bridge short circuit. As with the faulted speed response for the Level 4 Inverter, shown in Figure 10, the Permanent Magnet Machine accelerates in the negative direction after the fault is triggered due to the load torque acting as a mechanical source.

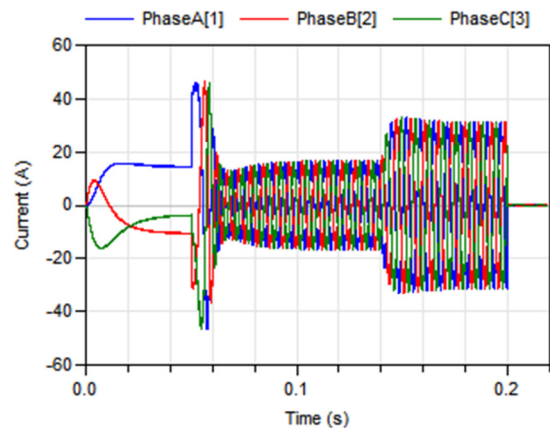


Figure 20: 3 phase current response with full-bridge open circuit fault at 0.2s

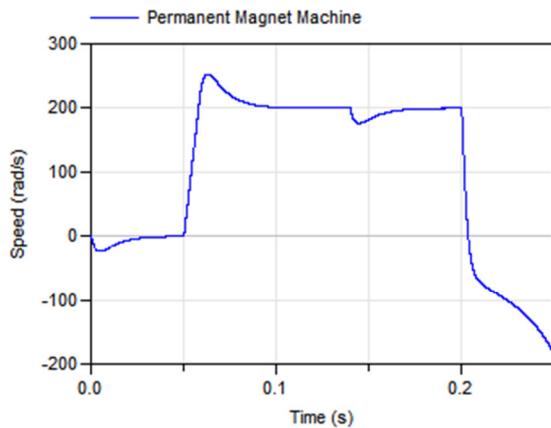


Figure 21: Speed response with short circuit at 0.2s

Overall this section has shown how there are a wide range of fault conditions which can be easily introduced and investigated using the Level 5 Inverter. This again emphasises the flexibility of the Inverters library presented here.

### 6.3 Losses

The Level 5 Switching model includes the conduction losses of the switch and when conducting. Energy losses due to switching are neglected. As can be seen from Figure 15 the FaultSwitch sub-component contains a HeatPort which outputs the heat flow generated from the switch and diode conduction losses to the Switching model. These outputs are then all connected to the inputs of a ThermalConductor and HeatCapacitor.

### 6.4 Thermal response

The Level 5 Switching model uses the MSL ThermalConductor, HeatCapacitor and PartialConditionalHeatPort as detailed for Levels 1 to 4 within Section 5.4. No PrescribedHeatFlow model is needed as the switch and diode conduction losses are already in the form of heat flow. However a HeatFlowSensor is included to give the user a numerical representation of the losses within the Level 5 Inverter.

## 7 Conclusion

This paper has presented a newly developed Power Electronic Inverter library in Modelica. This library utilises a multi-level approach which gives a high level of flexibility according to the user's needs. All levels are fully interchangeable and provide multi-directional power flow. In addition this new library gives the user the ability to include multi-directional

losses at different levels of accuracy while also providing the ability to introduce a multitude of Power Electronic Inverter fault conditions.

Overall this library is an extremely flexible multi-level tool which provides Power Electronic Inverter models that can be easily inserted, parameterised, interchanged and adapted to the user's requirements.

## 8 Future

At present the whole Actuation 2015 Actuator library, which includes this Inverters library, is in the process of experimental verification. The future availability and licensing of this library is also currently under discussion within the consortium.

## References

- [1] Assadian, F.; Fekri, S.; Hancock, M., "Hybrid electric vehicles challenges: Strategies for advanced engine speed control," *International Electric Vehicle Conference (IEVC)*, 2012.
- [2] Naayagi, R.T., "A review of more electric aircraft technology," *Energy Efficient Technologies for Sustainability (ICEETS)*, 2013.
- [3] A. Boglietti et al. "The safety critical electric machines and drives in the more electric aircraft: A survey", in *IEEE Industrial Electronics Conference*, 2009.
- [4] J. V. Gragger, et al. "The SmartElectricDrives Library – powerful models for fast simulations of electric drives," *Proceedings of the 5th International Modelica Conference*, pp. 571–577, 2006.
- [5] R. Rai, et al. "Simulation-Based Design of Aircraft Electrical Power Systems." *Proceedings of the 8th Modelica Conference*. 2011.
- [6] Funded by the European Commission within the Seventh Framework Program under grant FP7-284916 <http://www.actuation2015.eu/>
- [7] Van der Linden, F., Schlegel, C., Christmann, M., Regula, G., Hill, C.I., Giangrande, P., Mare, J.C., Egaña, I. "Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters." *Proceedings of the 10th International Modelica Conference*. 2014.
- [8] Van der Linden, F. "General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. *Proceedings of the 10th Modelica conference*. 2014.

# Mixed phasor and time domain modelling of AC networks with changeover management

Hakan Dogan Parildar\*, Alberto Leva†

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria  
Via Ponzio 34/5, 20133, Milano, Italy

## Abstract

Simulation studies on AC electric networks may comprehend periods of quasi-stationary operation and rapid transients. The adoption of a phasor-based approach results in high simulation efficiency, but is limited to the first of the two situations above, while for the second, time domain models are required. For system studies where both situations have to be simulated, a modelling paradigm is thus required that can join the two approaches in all the described components, and by which the simulator of an entire network can be endowed with the capability of moving back and forth from a phasor to a time domain system description automatically, taking care of the proper re-initialisations when necessary. In this paper we propose a possible solution, and apply our ideas by presenting the first *nucleus* of a Modelica library designed along them. We also show some simulation examples to support the validity and practical convenience of the proposal.

*Keywords:* AC networks, phasor models, time domain models, efficient simulation.

## 1 Introduction

Dynamic simulation is nowadays of great importance for both the design and the operation of electric networks [5]. Indeed, the availability of reliable simulation models is often enabling for the realisation of several functionalities of “smart” grids, like those described in works such as [9].

When addressing system studies – for example, but in principle not exclusively, for control synthesis purposes – it is frequently necessary to simulate long periods of (quasi-)stationary operation, interspersed with abrupt events. These two situations allow (and in some

sense call) for different modelling paradigms, however. Stationary operation is well described by phasor models, to the advantage of simulation efficiency. Events like for example the opening of a switch, on the other hand, require time-domain modelling, which is far more intensive from the computational standpoint. An intermediate case is given by “quasi-stationary” regimes, where the phasor approach can be extended by means of the so-called “swing equation”, which (simplifying for brevity) turns the algebraic phasor framework to a dynamic one, introducing machine angles as the state variables.

Focusing on the two extreme cases above, it would be highly beneficial and desired that a network simulation model could switch back and forth between a phasor and a time domain description, possibly in an automatic manner, and requiring as small an effort as possible on the part of the analyst. Quite expectedly, a significant research effort is being spent on the matter, but nonetheless some questions are still open, especially if a strict application of the object-oriented paradigm is required—i.e., more specifically, if all the encountered modelling issues are to be managed at the level of the individual components.

In this paper, we formulate a proposal to address the problem just mentioned, and prove its viability by presenting the first *nucleus* of a Modelica library, combining time domain and phasor modelling, developed along the devised approach.

The paper is organised as follows. Section 2 briefly discusses some related work, also to motivate the presented research, while Section 3 introduces the proposed approach, focusing on the automatic changeover problem. In Section 4 the library *nucleus* is outlined, and Section 5 reports a simulation example to demonstrate the viability and usefulness of the approach. Section 7 points out and synthetically discusses some open issues, while Section 8 draws some conclusions and sketches out future research.

\*Former graduated student at the Dipartimento di Elettronica, Informazione e Bioingegneria

†Corresponding author, [alberto.leva@polimi.it](mailto:alberto.leva@polimi.it)

## 2 Literature review and motivation

The need for modelling in the context of (AC) electric networks and their management has been recognised since long time ago [6]. The introduction of renewable energies and distributed generation has then increased the interest on the matter in the last decades [10], and further impulse to the mentioned research has been coming from smart grids [8].

Nowadays, simulation models of electric networks are also often combined with those of the generators' prime movers [1] to form multi-physics, multi-scale and potentially large overall models, for which the object-oriented paradigm of Modelica is particularly suited. In such cases, however, the electric part of the model is often the bottleneck for simulation efficiency, and the reason for that is structural.

In extreme synthesis, in fact, an AC network can be modelled at three levels. The most efficient one from the computational standpoint is provided by phasors [2]: this framework allows to write an algebraic model, that however is valid only in the hypothesis of a single, constant frequency for all the network. Small fluctuations of "local" frequencies are allowed by the so-called "swing equation" formalism, see e.g. [7], the application of which leads to dynamic component models, having machine angles as state variables.

Quite intuitively, the idea of using phasor models in Modelica is not new, but to the best of the authors' knowledge, to date no attempt was made to have phasor and time domain descriptions *co-exist* at the *component* level. For example, in [3] the idea of coupling phasor-domain and "transient" models is introduced, but the connection between the two relies on causal signals, making it difficult to represent it at the individual component level, especially for what concerns the domain changeover. Another interesting paper is [4], where however no changeover to time domain is considered, and the possibilities of phasor-based modelling are exploited via a convenient use of the swing equation.

As such, even a minimal literature analysis like that reported indicates that the problem addressed herein is of both theoretical and practical interest, and that the attempted solution has some novelty characteristics.

## 3 Mixed time-domain and phasor modelling

As anticipated, phasor models in Modelica are not a novelty [3, 4], and as such, the proposals that we are

making with this paper are to be integrated in the *scenario* depicted by works like the ones just quoted.

However, this research has some specific peculiarities, a discussion on which (and the consequently proposed solution) provide the main contributions of this paper. Specifically, three aspects are herein addressed:

- creating models that contain *both* a phasor and a time domain description of a given component,
- managing the transition (changeover) between the two at the component level,
- managing the *decision* to make a changeover at the overall model level.

As can be easily guessed, the main problem in structuring the required models is to preserve object orientation, the benefits of which are apparently not to be justified here, despite some facts to be handled are not confined to a single component—most notable, in this respect, is the changeover decision.

The rest of this section, organised along the items above, presents the proposed solution and provides motivations for it, also in a view to stimulating further discussions and improvements.

### 3.1 Component-level modelling

We now consider the problem of mixing phasor and time domain descriptions in the same component model, and of managing at that level the changeover between the two.

First, suitable physical (i.e., in this case, electrical) connectors are necessary. To this end, a positive phasor and time domain pin is straightforwardly defined (we omit trivial code like the inclusion of the `Modelica.SIunits` package) as

```
connector ptPin "phasor/time positive
  pin "
    Voltage Vre "V phasor , R part";
    Voltage Vim "V phasor , I part";
    Voltage v "v(t), time domain";
  flow Current Ire "I phasor , R part";
  flow Current Iim "I phasor , I part";
  flow Current i "i(t), time domain";
end ptPin;
```

Listing 1: connectors.

and a negative one is created in the same way. Then, each component has to contain the constitutive equations for both the phasor and the time domain description, and to this end, we have to distinguish between passive components and active ones—i.e., generators.

Taking the Inductor component as a representative example for the former type, and denoting by  $a$  and  $b$  its positive and negative pin, respectively, the time domain description is

$$\begin{aligned} 0 &= a.i + b.i; \\ a.v - b.v &= L * \text{der}(a.i); \end{aligned}$$

Listing 2: inductor, time domain equations.

while the phasor one reads

$$\begin{aligned} 0 &= a.Ire + b.Ire; \\ 0 &= a.Iim + b.Iim; \\ a.Vre - b.Vre &= (L * \text{freqHz} * 2 * \pi) \\ &\quad * \text{sqrt}(a.Ire^2 + a.Iim^2) \\ &\quad * \text{cos}(\text{atan2}(a.Iim, a.Ire) \\ &\quad \quad + \pi / 2); \\ a.Vim - b.Vim &= (L * \text{freqHz} * 2 * \pi) \\ &\quad * \text{sqrt}(a.Ire^2 + a.Iim^2) \\ &\quad * \text{sin}(\text{atan2}(a.Iim, a.Ire) \\ &\quad \quad + \pi / 2); \end{aligned}$$

Listing 3: inductor, phasor equations.

Finally, the code for initialising the time domain equation when needed is

```
if flag_for_reinit then
  when {TimeDom} then
    reinit(a.i,
      sqrt(pre(a.Iim)^2 + pre(a.Ire)^2)
      * sin(2*pi*freqHz*time
      + atan2(pre(a.Iim), pre(a.Ire)))
    end when;
end if;
```

Listing 4: inductor, time domain (re-)initialisation.

where `TimeDom` is a flag indicating that the entire model is simulated in the time domain.

Coming to generators, the relevant equations for an ideal sine voltage one are shown below.

```
0 = a.Ire + b.Ire;
0 = a.Iim + b.Iim;
cos(phase)*V = a.Vre - b.Vre "Re of V";
sin(phase)*V = a.Vim - b.Vim "Im of V";
0 = a.i + b.i;
if TimeDom then
  a.v - b.v = V * sin(2*pi*freqHz*time
    + phase);
else
  a.v - b.v = 0;
end if;
```

Listing 5: sine voltage generator equations.

At the component level, changeover is thus managed by acting in a coordinated manner on passive and active components, in a view to avoiding unnecessary events and enhance simulation speed. It can be in fact noticed that when the phasor model is in use, the time domain voltage is just zeroed. This makes the state variable `a.i` in Listing 2 simply decay to zero, avoiding conditional equations and limiting the triggered events to the bare minimum (the transition of `TimeDom` from true to false, or *vice versa*). Of course, said decay to zero is meaningless as simulation result, which is however indicated by the value of `TimeDom`.

The only *caveat* with the proposed solution is that the currents of inductors in series – or, dually, the voltages of capacitors in parallel – may be treated by the tool at hand as a single entity. The parameter `flag_for_reinit` in Listing 4 has the purpose of avoiding inconsistencies in such cases. At present, it is the user's responsibility to ensure that in each set of inductors in series, or capacitors in parallel, one and only one has that parameter set to true. Apart from this, however, object orientation (or more specifically, the independence of a component description of how it is connected to the others) is fully preserved.

## 3.2 System-level modelling

We now move to the problem of managing the changeover between the time domain and the phasor representation at the level of the entire simulation model, i.e., of the network—in synthesis, thus, we specify how the `TimeDom` flag is handled.

Switching from phasor to time domain is generally the consequence of some abrupt event known to at least one component, like for example a closing or opening switch. It is thus assumed that in such a case, the affected component directly sets the flag to true, causing the changeover instantaneously.

A bit more difficult is conversely the reverse changeover, since to make it feasible, *all* the currents and voltages need settling to a sinusoidal regime. The decision is in this case taken on the basis of local signalling from the components, and of a unanimity verification mechanism at the system level.

### 3.2.1 Local signalling of sinusoidal regime

Also in this case, like it was for the component-level changeover management shown in Section 3.1, the main goal of model design is to avoid unnecessary events, for efficiency reasons.

Suppose therefore that we need to detect if a certain variable  $x(t)$  – voltage or current – is “sufficiently” close to a sinusoid with frequency  $f \text{ reqHz}$ , assumed for the moment of zero mean for simplicity (releasing this is straightforward, see later on). Filtering  $x(t)$  through the continuous-time transfer function

$$F(s) := \frac{X_F(s)}{X(s)} = \left( \frac{1}{\pi f_o} \frac{s}{1 + \frac{s}{\pi f_o} + \frac{s^2}{(2\pi f_o)^2}} \right)^{n_F} \quad (1)$$

produces an output  $x_F(t)$  equal to the input  $x(t)$  if and only if the frequency of the latter is exactly  $f_o$ , which is apparently set to  $f \text{ reqHz}$ .

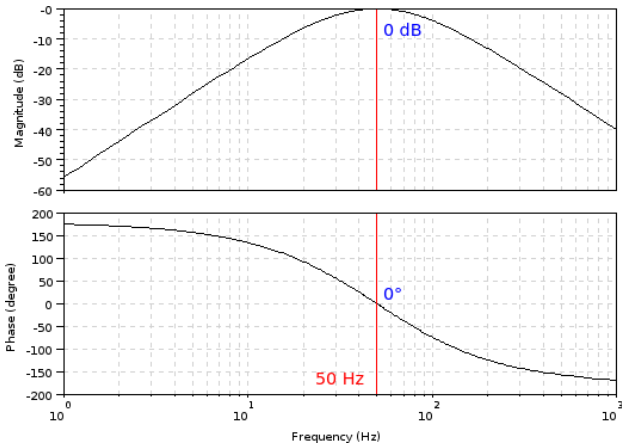


Figure 1: frequency detector – Bode plots of  $F(j\omega)$ .

Parameter  $n_F$  is used to enhance the attenuation of the frequency response  $F(j\omega)$  as the input frequency moves away from  $f_o$ , and a value of two (or three at most) proved enough in practice. The operation of  $F(s)$  is shown by the Bode plots of Figure 1, obtained with  $f_o = 50 \text{ Hz}$  and  $n_F = 2$ .

The output of  $F(s)$  in (1) is then used, together with its input, to form the signal

$$s(t) = \frac{1 + x_f(t)^2}{1 + x(t)^2} \quad (2)$$

which is structured so as to inherently avoid division by zero errors, and finally  $s(t)$  is lowpass-filtered by the unity-gain first-order block

$$D(s) := \frac{Y(s)}{S(s)} = \frac{1}{1 + s \frac{k_f}{2\pi f_o}} \quad (3)$$

where parameter  $k_f$  is used to control the achieved smoothing (a value of ten is a good default). As a result,  $y(t)$  will signal the required condition on  $x(t)$  by taking a value very close to the unity, with small fluctuations.

Comparing the value and the time derivative of  $y^2(t)$  to suitable thresholds, where squaring the signal is to avoid the events that would be generated if its absolute value were conversely taken, is therefore a means to detect that  $x(t)$  is close enough to a sine wave with frequency  $f \text{ reqHz}$ .

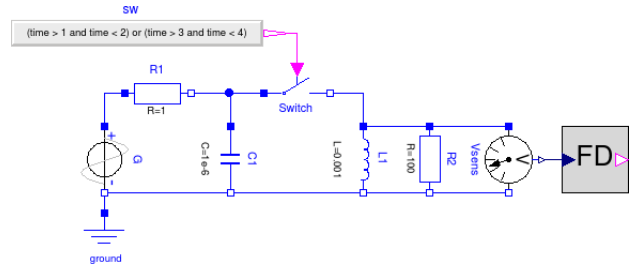


Figure 2: frequency detector test – Modelica diagram.

To show the efficacy of the proposed technique, and also its autonomy with respect to the rest of the proposed modelling paradigm, Equations (1) through (3), together with the mentioned thresholding mechanism, were turned into a `FreqDetector` block, that is used in the model of Figure 2 together with components from the Modelica.Electrical library (its use in the presented one, with the actual introduction of phasor modelling, will be illustrated later on).

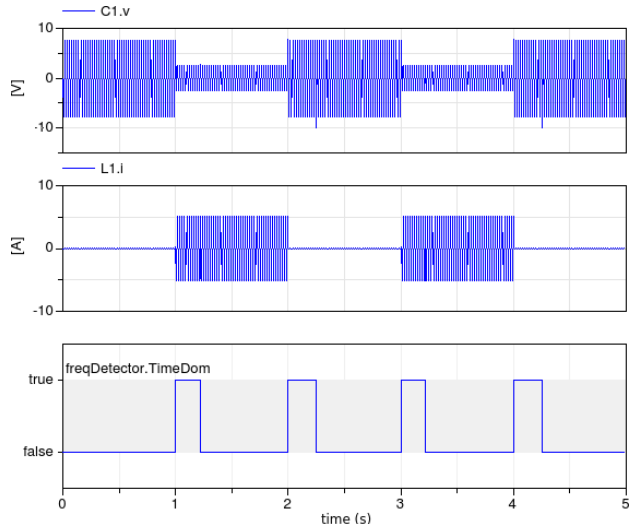


Figure 3: frequency detector test – simulation results.

Figure 3 shows a sample simulation test. Detailed figures are inessential for its purpose, apparently, but as can be seen, the need for time domain modelling as caused by the switch closing and opening is detected correctly, especially for the transition toward (the possible use of) phasor mode.

Recall that the symmetric transition is in any case guaranteed by the locally originated signalling, see



Section 3.1 above. This is because the phasor to time domain transition must be *instantaneous* to preserve accuracy, while if the time to phasor domain is delayed with respect to the time when it is acceptable, the only relevant effect is some waste of CPU time.

It is worth noticing that the 5s simulation of Figure 3 involves only 14 state events, which would apparently not be true if the possibility of switching to phasor mode were identified based on zero crossing counters, or similar methods. Note also that the proposed technique introduces some additional state variables, but these pertain to *linear, time invariant, causal* blocks *cascaded* to the physical model. The resulting simulation overhead is thus modest, and in any case much lower than that of zero-crossing or similar methods.

The default values chosen allow for a safe transition toward phasor modelling, without bounces and within a reasonable time. In the presence of an essentially constant-frequency behaviour interspersed with abrupt events that make the frequency content of the involved signals radically different from the (quasi-)stationary one, this is a good compromise between fast transition to phasor mode (which undoubtedly favours simulation efficiency) and possibly undue switching of the two modes (which conversely may be detrimental owing to re-initialisations). Also, the presented detection method is inherently normalised, since so are all the involved quantities (except times, of course). This makes the selected default values for the involved parameters valid in a wide operating range, and for an equally wide variety of network physical parameters.

To manage a possible nonzero average of  $u(t)$ , finally, the proposed filtering path is implemented as a series of transfer function blocks from the Modelica Standard Library, and signal  $s(t)$  in (2) is formed by taking the output of the first block in the place of  $u(t)$ . This ensures that the average of the signal taken as input settles to zero with a dynamics comparable to that of the transients superimposed to its steady-state sinusoidal behaviour, and at the same time preserves the exploitation of the unity-magnitude and zero-phase frequency response values at the sought frequency so as to realise the envisaged detection system.

### 3.2.2 System-level handling of TimeDom

To manage the TimeDom flag at simulation time, denote respectively with  $N_{pt}$  and  $N_{tp}$  the number of elements entitled to cause a changeover toward time domain mode (typically, switches), and that of elements the voltage across which is to be checked to approve the reverse transition.

From a conceptual standpoint this set could well be the totality of the present passive components, but for optimisation reasons the user can be allowed to introduce “frequency probes”, based on the described FreqDetector component, only where deemed necessary.

At the present state of the library development, this architectural choice is still open: most likely, however, based on the experience that is being gathered, the final solution will be to distinguish a “basic” mode, where any passive component has a detector, and an “expert” one, where the user is free to configure the mechanism at his/her best convenience.

In any case, assuming – in principle, as the implementation described in the following section is different for efficiency reasons – two boolean vectors P2T and T2P, of length  $N_{pt}$  and  $N_{tp}$  respectively, to be declared at the outermost model level, and omitting trivial details on the inner/outer manner they are managed, the following procedure for handling TimeDom is adopted.

1. The simulation starts out in time domain mode, for the safe side (possibly, in the future, unless differently specified in the “expert” mode). All the elements of P2T and T2P are conveniently initialised (at present, to false).
2. All entitled elements manage their local time domain flag based on the contained FreqDetector element, and the system-level T2P vector collects them all.
3. If at the system level time domain mode is in use, and all the elements of T2P are true, then the system switches to phasor domain mode.
4. If at the system level phasor mode is in use, any transition to true of at least one element of P2T causes a changeover to time domain mode, with the required re-initialisations.

### 3.2.3 Modelica implementation

The solution just described serves the intended purpose, and the realised one is totally equivalent from a conceptual and functional standpoint.

However, if said solution were implemented literally, some deviation from a totally object-oriented setting would be involved, since any component participating in either of the two mentioned boolean vectors, would have to contain suitable parameters to indicate

which position in said vectors pertain to it. The responsibility of setting those indices correctly would stand with the user, being possibly complex and cumbersome to manage for large models.

In addition, and most important, even if the management of the mentioned indices were somehow automated, some model connections would in this way be realised, that do not fall under a proper connector abstraction.

To overcome this relevant problem, the described solution is therefore implemented as follows. First, a `ChangeoverMgmt` connector is defined as shown below.

```
connector ChangeoverMgmt
flow Integer Ntp "# of T->P voters";
flow Real ForceP2T;
flow Real AllowT2P;
Modelica.SIunits.Frequency freqHz;
Boolean TimeDom;
Real dum "Squelch balancing warnings";
end ChangeoverMgmt;
```

Listing 6: the `ChangeoverMgmt` connector.

Each component participating in the changeover decision (i.e., each model of reactive elements or of commuting ones like switches), and also each generator, is endowed with such a connector, named in the following `C`; also, all those models take the `TimeDom` flag and the (nominal) frequency from that connector.

Reactive components (the *Inductor* model is an example) furthermore contain the code

```
C.Ntp      = -1;
C.ForceP2T = 0;
...
C.AllowT2P = if C.TimeDom and not
             FD.TimeDom
             then -1 else 0;
```

Listing 7: inductor, changeover management.

The first line reported in Listing 7 provides the `Supervisor` component, described later on and that also has a `ChangeoverMgmt` connector to which those of all network elements are connected, with the number of those elements that vote for the time domain to phasor mode transition.

The second line means that the component, given its role, is not entitled to force a transition from phasor to time domain model.

The last reported line, finally, casts the vote when this is required.

Models of commuting components (like switches) conversely contain the code

```
parameter Real Tsw    = 0.01;
parameter Real thrsw  = 0.01;
...
Real xsw(start=1);
...
xsw+Tsw*der(xsw) = if control
                   then 1 else 0;
C.ForceP2T      = if control
                   and xsw<1-thrsw
                   or not control
                   and xsw>thrsw
                   then -1 else 0;
```

Listing 8: inductor, changeover management.

When the control input (the switch command) commutes, the introduced dynamic variable `xsw` is used to generate a square pulse with a minimum of state events, and this is in turn used – see the last line in Listing 8 – to signal that the component intends to force a transition from phasor to time domain mode.

Finally, the *Supervisor* component is implemented as per Listing 9 below.

```
model Supervisor
parameter Frequency fo=50;
Interfaces.ChangeoverMgmt C;
Boolean P2T, T2P;
equation
C.dum      = 0;
P2T        = C.ForceP2T>0.5;
T2P        = C.AllowT2P>C.Ntp-0.5;
C.freqHz   = fo;
algorithm
when T2P and not P2T then
    C.TimeDom := false;
end when;
when P2T then
    C.TimeDom := true;
end when;
initial equation
C.TimeDom = true;
end Supervisor;
```

Listing 9: supervisor.

The initial equation makes the simulation start in time domain, as specified in Section 3.2.2. Then, thanks to the flow connections, variables `ForceP2T` and `AllowT2P` respectively sum the forcing to time domain mode requests, and the permission for phasor mode votes. based on that, when `ForceP2T` is at least 0.5, then at least one component is forcing time domain mode.

Analogously, when AllowT2P exceeds the number of voters (collected in the Ntp connector variable) minus 0.5, then all said voters are permitting the transition toward phasor mode. Finally, the two when clauses in the algorithm section manage the TimeDom flag, triggering events only when necessary.

### 4 The library structure

As anticipated, the presented ideas were applied to create the first *nucleus* of a Modelica library for mixed phasor and time domain modelling of electric networks.

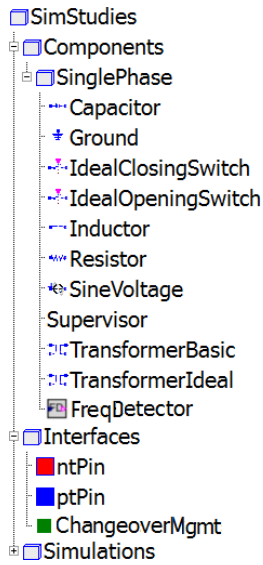


Figure 4: the library (*nucleus*) structure.

Said *nucleus* is structured as outlined in Figure 4, and to date comprises only single-phase components, connectors, and the Supervisor block to be included in the realised models so as to implement the necessary network-wide declarations, and the management of the involved variables.

Plans are of course to extend the *nucleus* to form a complete library, including more articulated components such as non ideal generators, the representation of multiphase elements, and the like. Also, care will be taken to integrate the consequent future work with (at least) the similar ones shown in the references quoted above in Section 2.

### 5 An illustrative simulation example

We now shows a simple simulation example to demonstrate the operation of the proposed modelling framework, and specifically of the changeover management.

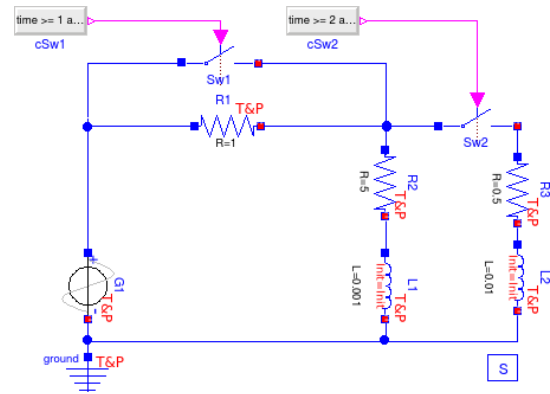


Figure 5: illustrative simulation example – the used network model.

The example refers to the small network model depicted in Figure 5. The switch  $Sw1$ , initially closed, is opened at  $t = 1\text{ s}$  and re-closed at  $t = 3\text{ s}$ . The switch  $Sw2$ , also initially closed, is cycled at  $t = 2\text{ s}$  and  $t = 4\text{ s}$ . The simulation run duration is  $5\text{ s}$ .

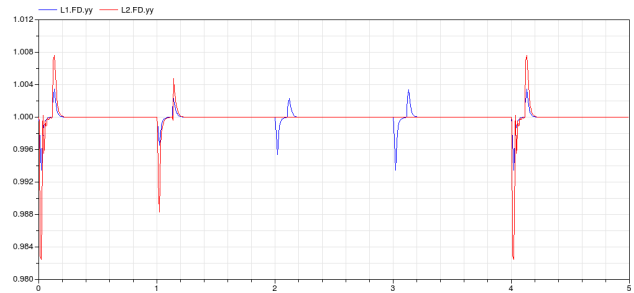


Figure 6: illustrative simulation example –  $y$  signals in the frequency detectors of the two inductors.

Figure 6 shows the  $y(t)$  variables – see (3) – in the frequency detectors of the two inductors. Recall that those variables are meant to indicate, by assuming a nearly constant unity value, the settling of the locally measured frequency to  $\text{freqHz}$  (here set to  $50\text{ Hz}$ ). As can be seen, the expected effect is obtained, and the normalised nature of the involved signals produces comparable transients also in the presence of different values for the components’ electric parameters.

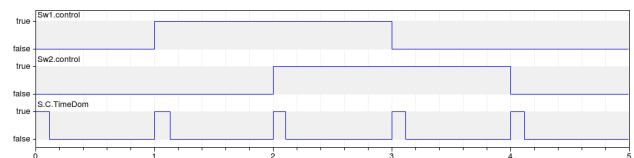


Figure 7: illustrative simulation example – boolean flags.

In Figure 7, the relevant boolean flags at the system level are instead represented. The changeover mecha-

nism catches the switch events, passing to time domain instantaneously, while the time domain periods are not equal, correctly depending on the transient behaviour of the monitored electric variables.

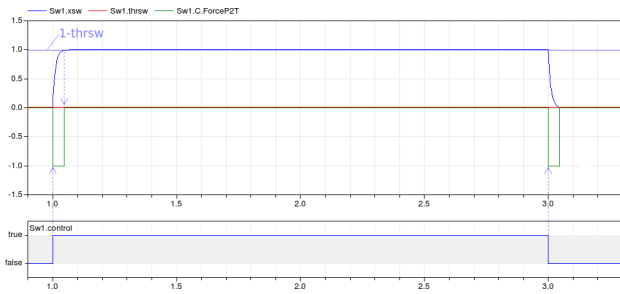


Figure 8: illustrative simulation example – forcing time domain on the part of a switch.

Figure 8 illustrates how a commuting component forces the transition to time domain mode by means of `xsd`, and the role of the threshold and the time constant of its dynamics in determining the width of the generated pulse. Note that the rising edge of that pulse is structurally synchronous to the switch command, which is consistent with the defined specifications.

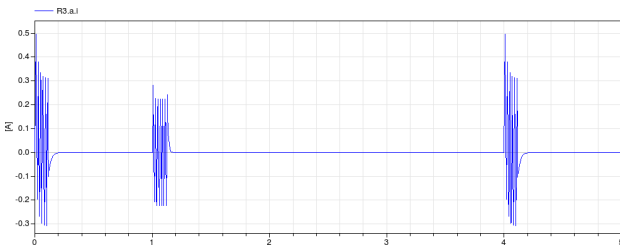


Figure 9: illustrative simulation example – current in R3, time domain representation.

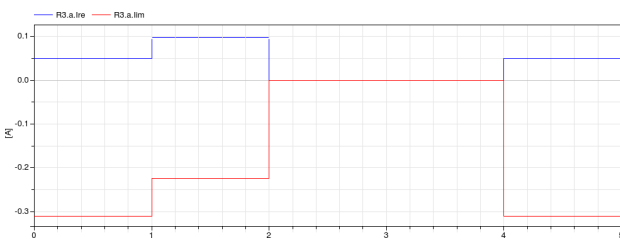


Figure 10: illustrative simulation example – current in R3, phasor representation.

Figures 9 and 10 show the behaviour of the current flowing through resistor R3, viewed respectively in the time domain and as a phasor (for which the real and the imaginary part are plotted). Notice that when the simulation switches back to phasor mode after a period in time domain mode triggered by a switch event,

the time domain signal has actually settled back to a sinusoidal behaviour, making the changeover correct.

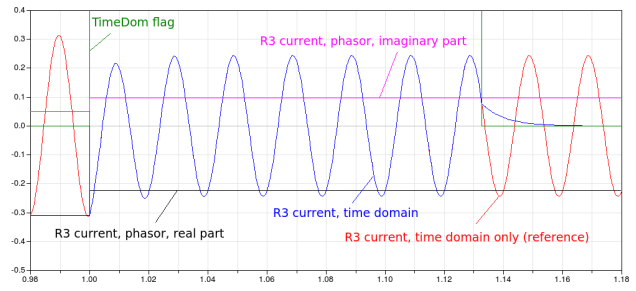


Figure 11: illustrative simulation example – current in R3, changeover detail.

Finally, Figure 11 demonstrates the changeover and re-initialisation mechanism, comparing the phasor and time domain representations for the current through R3 with a reference obtained by forcing the simulation to take place entirely in the time domain. Observe the (re-)initialisation of the time domain representation of the shown variable, triggered instantaneously by the changeover to time domain mode. Note also how the time domain representation of the same variable settles to zero after the changeover to phasor mode occurs, allowing the solver to exploit variable-step capabilities to the advantage of efficiency. As can be verified, then, taking the time or phasor domain signals as the valid ones depending on `TimeDom`, conveys all the required information.

## 6 A simulation example on efficiency

Although on this matter work is still in progress, we report an example showing the simulation efficiency gained with the proposed approach.

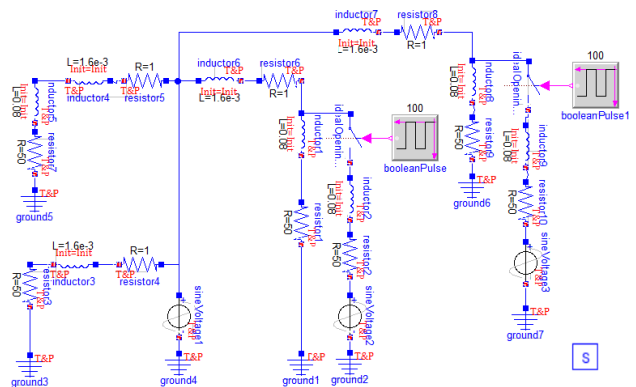


Figure 12: simulation example on efficiency – the used network model.

The example refers to the network model depicted

in Figure 12, apparently more complex than that of Figure 5 above. The network was assembled with components from the presented library, and a reference model for it was created with the equivalents from the Modelica Standard Library, limiting of course the scope to time domain modelling. Various simulations were then performed with a duration of  $10^4$  seconds, cycling the switches at regular intervals for a variable number of times.

Cycles	CPU time	Steps	F-evals	J-evals
800	161	26276873	52580944	16847
600	155	26038476	52097092	12594
400	164	26036633	52086819	8404
200	161	26103648	52214194	4259
100	160	26177418	52358292	2131
80	168	26095949	52194637	1699
40	162	26126749	52254882	853

Table 1: simulation example on efficiency – results with the Modelica Standard Library.

Cycles	CPU time	Steps	F-evals	J-evals	Events
800	134.00	1717702	5968550	759422	59257
600	97.60	1246309	4293129	538653	42139
400	67.80	857706	2980426	379132	29580
200	33.70	429941	1493272	189871	14758
100	16.90	215300	747528	95007	7424
80	13.90	173998	605417	77192	6008
40	7.12	90491	316665	40846	3207

Table 2: simulation example on efficiency – results with the proposed models.

Table 1 shows the results obtained with the Modelica Standard Library. The first column reports the number of switch cycles in the simulation, while the others contain the typical computational load statistics. Table 2 conversely shows the results achieved with the proposed models, and has a further column that reports the number of generated state events (that apparently is zero in the previous case).

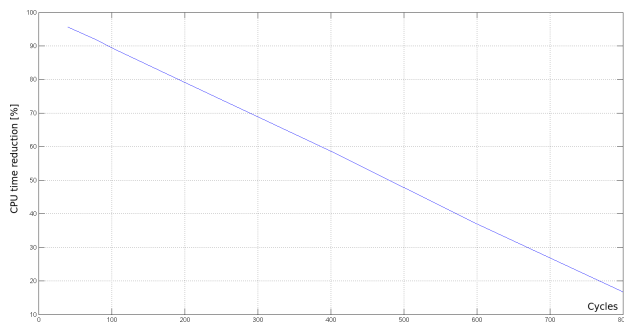


Figure 13: simulation example on efficiency – CPU time percent reduction versus switch cycles.

As can be seen, the efficiency improvement is quite evident in terms of CPU time, for which the gain is represented graphically in Figure 13, number of steps and F-evaluations, although (owing most likely to the numerous re-initialisation) there is quite an increase of the Jacobian evaluations.

## 7 Open issues

As stated right from the introduction, the work presented in this paper is still in progress. We have reached sufficient certainties on the viability of the approach, with particular reference to the possibility of managing the representation changeover within a solid object-oriented setting, and without triggering undue event hauls. However, more than one relevant issue stands still open, and this section is devoted to a brief overview on them.

A first point concerns the quantification of the obtained advantages. Here some figures referring to efficiency were given, but no doubt, more extensive efficiency assessment campaigns are in order, and to this end, testing is underway with larger models. At a first glance, the impression we are getting is that the number of simulation steps reduces more or less proportionally to the fraction of the overall simulation run that can be done with phasors, which is in reasonable accordance with intuition. On the other hand, the number of Jacobian evaluation exhibits more peculiar behaviours, on which deeper investigations are also to be performed. Also, it is advisable that future studies address possible interactions with the solution algorithm, of the variable-step type for apparent reasons: at present, in fact, to date all the tests were done with DASSL only.

A second point concerns the parameters involved in the changeover management machinery. Despite the use of normalised signals has proven effective, in fact, we still observe some residual effects of the threshold values, on which further studies are consequently in order. In particular, it would be useful to somehow relate said thresholds, and also the value of the time constant for the dynamics of *x<sub>sd</sub>*, to the desired tradeoff between accuracy and efficiency improvement, quantified (for example) on the basis of the additional time spent simulating in time domain mode beyond necessary.

Moving from efficiency to model manipulation and structuring issues, a first point concerns the re-initialisation flags. At present these are to be managed by the user, but apparently could be set automati-

cally if the tool manipulation chain were somehow instructed to recognise and treat series/parallel connections. In principle, leaving the flag responsibility to the user does not seem a big problem, unless when dealing with large models where connections are changed frequently—not too realistic a case. Nonetheless, also on this point further research is advisable.

A more relevant issue is relative to the use and abstraction of non-physical connectors like the `ChangeoverMgmt` one. Here, the quest for efficiency in fact led to somehow abuse flow variables (by the way, the authors would like to thank the colleagues Francesco Casella and Victorino Sanz for ideas and useful suggestions in this respect). However, with the mechanism here adopted, more articulated voting mechanisms than the realised ones, would be cumbersome and possibly impossible to implement. In contexts like that addressed herein such mechanisms may not be necessary, but the experience reported in this work suggests that some way to tailor the connector semantics would be highly desirable—a small suggestion for discussions on future versions of the language.

Finally, although this is undoubtedly simpler and less important from a conceptual point of view, some post-processing tool would be needed that uses the produced results (phasors and time domain variables, together with `TimeDom` to say which one is significant at which time instant) to reproduce and present all the simulated quantities entirely in the time domain, for the user convenience.

## 8 Conclusions and future work

The problem of creating models of AC networks joining the phasor and the time domain approach, and switching from one another automatically during a simulation run, was addressed. A solution was proposed, having some peculiar features in particular as for the changeover mechanism, and motivations for the adopted choices were provided. The approach was put to work by creating the first *nucleus* of a Modelica library, and a simulation example – referring to a very simple case given the purpose of this paper – was shown to back up the viability and usefulness of the proposal.

Future work will be essentially aimed at tackling the issues pointed out in Section 7. In addition, the integration with models for the process sections of generators (typically, their prime movers) will be addressed, in a view to better exploit the efficiency improvements yielded by the proposed approach. Finally, given the

interest on the considered modelling and simulation domain, the authors hope that this proposal can stimulate discussions, further ideas, and collaborations.

## References

- [1] F. Casella and A. Leva. Modelling of thermo-hydraulic power generation processes using Modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12(1):19–33, 2006.
- [2] C.A. Desoer and E.S. Kuh. *Basic circuit theory*. McGraw-Hill, New York, NY, 1966.
- [3] O. Enge, C. Clauß, P. Schneider, P. Schwarz, M. Vetter, and S. Schwunk. Quasi-stationary AC analysis using phasor description with Modelica. In *Proc. 5th International Modelica Conference*, pages 579–588, Vienna, Austria, 2006.
- [4] A. Haumer, C. Kral, J.V. Gragger, and H. Kapeller. Quasi-stationary modeling and simulation of electrical circuits using complex phasors. In *Proc. 6th International Modelica Conference*, pages 229–236, Bielefeld, Germany, 2008.
- [5] W.W. Hogan. Markets in real electric networks require reactive prices. In M. Einhorn and R. Siddiqi, editors, *Electricity transmission pricing and technology*, pages 143–182. Springer, Dordrecht, The Netherlands, 1996.
- [6] M. Huneault and F.D. Galiana. A survey of the optimal power flow literature. *IEEE Transactions on Power Systems*, 6(2):762–770, 1991.
- [7] D.P. Kothari. *Modern power system analysis*. Tata McGraw-Hill Education, Noida, India, 2003.
- [8] V. Liberatore and A. Al-Hammouri. Smart grid communication and co-simulation. In *Proc. Energytech*, pages 1–5, 2011.
- [9] J.A.P. Lopes, C.L. Moreira, and A.G. Madureira. Defining control strategies for microgrids islanded operation. *IEEE Transactions on Power Systems*, 21(2):916–924, 2006.
- [10] J.R. Ubeda and M.A.R. Rodriguez Garcia. Reliability and production assessment of wind energy production connected to the electric network supply. In *IEE Proceedings – Generation, Transmission and Distribution*, volume 146, pages 169–175, 1999.

# impact – A Modelica<sup>®</sup> Package Manager

Michael Tiller  
Xogeny Inc., USA  
michael.tiller@xogeny.com

Dietmar Winkler  
Telemark University College, Norway  
dietmar.winkler@hit.no

## Abstract

To manage complexity, modern programming languages use organizational units to group code related by some common purpose. Depending on the programming language, these units might be called libraries, packages or modules. But they all attempt to encapsulate functionality to promote modular code and reusability. For the remainder of this paper, we will simply refer to these organizational units as *packages* (as they are called in Modelica).

Also common to many modern programming languages are tools to manage these packages. These tools are generally called *package managers* and they allow developers to quickly “fetch” any packages they may need for a given project. The main functions of package managers are to allow developers to search, install, update and uninstall packages with a simple command-line or graphical interface. In the Java world, the most common package manager is maven. For Python, tools like `easy_install`[1] and `pip`[2] are used for managing packages. For client-side web development, `bower` is used. For server-side JavaScript, the tool of choice is `npm`[3]. For compiled languages, these package managers often include some additional build functionality as well.

This paper introduces *impact*, a package manager for Modelica. Using *impact*, Modelica users and developers can quickly search for, install and update Modelica libraries. In this paper, we will discuss the functionality provided by *impact*. In addition, we will discuss how the functionality was implemented. As part of this we will discuss the importance of collaborative platforms, like GitHub[4] in our case, for providing a means for collecting, curating and distributing packages within a community of developers.

The *impact* package manager is provided to the Modelica community as a free, open-source tool.

Furthermore, the protocols involved are all documented and we encourage tool vendors to integrate them into their own tools so they can provide the same searching, updating and installation capabilities that the command-line tool provides.

**Keywords:** *Modelica, package manager, GitHub, dependency management, Python*

## 1 Introduction

It is increasingly the case that the adoption of new technologies hinges on automating away the tedious tasks required to learn and adopt these new technologies. For programming languages or frameworks, this means streamlining the process by which libraries can be found and installed.

For nearly all modern languages, this issue of “package management” has reached the point where it is almost an element of language design. The Java world has the maven tool, Scala has `sbt`, Node.js has `npm` and the Go language includes built-in support for package management via its the command line `go` compiler.

In the Modelica world, this issue has been largely overlooked. Although there have been proposals for formats to list network accessible libraries, these efforts have remained mere proposals without any concrete functionality. The *impact* project was inspired by the Bower[5] project’s approach. This lightweight, `git`-centric approach (discussed in Section 3) turned out to be relatively straightforward to implement and provides functionality otherwise unavailable in the Modelica world.

The goal of the *impact* project is to provide the same basic package management features found in most package managers. These features will be dis-

cussed in detail in subsequent sections of this paper (see Section 2). Our goal is to lower the barrier for users to find, install and update libraries. At the same time, we expect that the `impact` tool itself will be just as easy to install as the libraries it supports.

The contribution of the `impact` project is making installation of Modelica packages as easy as possible. There are actually three important aspects in our approach. The first aspect is the one most apparent to the user, a command-line interface that can be used to easily install not just a given Modelica library, but **also its dependencies**. However, such a command line tool must rely on the second aspect which is the availability of a centrally served, up to date index of packages. The final aspect is making it easy for library developers to **publish** their libraries in such a way that they are available to other Modelica users through the `impact` package manager. Each of these aspects will be discussed as part of this paper.

It is worth noting that while `impact` can handle dependences, it does not solve some of the problems currently inherent in Modelica. At the moment, dependencies between packages are described by individual versions. The result is that these dependencies can create brittle chains which are not always possible to satisfy. The logic for unconverging dependencies in `impact` is very simple. It merely identifies any dependencies explicitly listed by the library and then attempts to find that version of those packages within the `impact` package index. Hopefully the Modelica annotations to express dependencies will be refined to support a richer set of relationships. If so, the logic used by `impact` to identify and install these dependencies can be extended to support this improved expressiveness.

## 2 Command Line Interface

### 2.1 Installation of `impact`

The `impact` tool is available from “PyPI”[6] and can be installed by running either

```
$ pip install impact
```

or

```
$ easy_install impact
```

As an alternative one can also download the sources from <https://github.com/xogeny/impact> or <https://pypi.python.org/pypi/impact>, unpack and run

```
$ python setup.py install
```

in order to install it.

### 2.2 Searching

Searching for libraries is done by executing:

```
$ impact search <search term>
```

This will print a list of all package whose names and/or description strings contain the “<search term>”. The returned list also contains the URL where the Modelica package is hosted.

The output can also be made more verbose with:

```
$ impact search -v <search term>
```

which, in addition, will return the description string and the available versions.

### 2.3 Installing packages

Once a package of interest is found using `search`, it can be installed by executing:

```
$ impact install <package name>
```

This will then fetch not only the package itself and extract it in a configurable target directory but **it will also fetch the dependencies of the packages** as long as those are available to `impact`.

If several versions of a package are available, `impact` will choose the latest one. If this is not desirable then one can also specify the version explicitly. For example, in order to install Modelica version 2.2.2 one would execute the command:

```
$ impact install Modelica#2.2.2
```



Just like for the `search` sub-command there is also a verbose switch “-v” available for `install` which will give information on what versions are available, which version is going to be installed, what the dependencies are and where the version will be downloaded from.

In addition there is also a “[-dry-run|-d]” option available which does not download or extract any files but will simply report what `impact` would do. Users will generally use the “dry-run” option in combination with the “verbose” option.

### 3 Candidate Packages

The question that might arise now is, how does `impact` know what packages and which versions of those packages are available.

#### 3.1 Making packages visible

The Modelica Association (MA) has always maintained a list of available Modelica libraries on their website under <https://modelica.org/libraries>. Initially, the list was a static web page which listed the different packages and their latest version as submitted to the webmasters of the MA. Keeping the list of packages up-to-date was a manual job for both the website maintainers and package developers.

In spring 2013, all the free packages listed on <https://modelica.org/libraries> were moved to individual repositories on GitHub. Third-party packages can be found under <https://github.com/modelica-3rdparty> and packages by the MA under <https://github.com/modelica>. This had the following benefits:

- Package developers can access their package repository directly without having to involve the webmasters of the MA thus submitting updates any time.
- All packages now have an individual issue tracker and version control service in place.
- The webmasters of the MA can now collect the latest information on all the packages automatically in order to generate an up-to-date list-

ing on <https://modelica.org/libraries> (more on this later in Section 3.3).

#### 3.2 Semantic versioning

One thing that is important when trying to build up a package manager that can also handle version dependencies is the need for a proper approach to version numbering.

We decided to base our package manager on a system called Semantic Versioning[7]. As a result, package developers are strongly encouraged to use semantic versioning so that they are compatible with `impact`. This has the additional benefit of being a well-documented and logical approach.

Semantic versioning has the simple rule of:

*Given a version number MAJOR.MINOR.PATCH, increment the:*

1. *MAJOR version when you make incompatible API changes,*
2. *MINOR version when you add functionality in a backwards-compatible manner, and*
3. *PATCH version when you make backwards-compatible bug fixes.*

In addition, pre-releases (*e.g.*, beta releases, release candidates) and build metadata (*e.g.*, version control hashes) are also taken care of in this system, details can be found in the manual page[7].

#### 3.3 GitHub API

Having all packages available as GitHub repositories means that we can use the *GitHub API v3*[8] in order to collect data about those packages. All API access is over HTTPS, and accessed from the `api.github.com` domain. All data is sent and received as JSON[9].

For example if one visits: <https://api.github.com/users/modelica/repos> a verbose list containing a series of information of all repositories that exist under the user `modelica` is returned in JSON format. This includes also a new API-url for retrieving the tags of a specific repository. For example, by visiting <https://api.github.com/repos/modelica/Modelica/tags> we get a list of

all tags for the Modelica Standard Library repository including download links for a zipped version of the tagged source code.

As mentioned before, all data is returned as JSON according to a proper schema. This makes it easy to pull out all the information we need. Initially, this information was used to generate an up-to-date listing of all MA and third-party packages to be displayed on <https://modelica.org/libraries> but we also noticed quite quickly the possibilities such an API opens up. It was offering the very information we needed in order to build a catalog of available packages including the different tagged versions for download.

### 3.4 GitHub only?

The mechanism described so far seems to depend a lot on GitHub’s API. So one might wonder is there a danger of locking us to GitHub.

The answer is actually no. We chose GitHub just as **one** possible data source. It is possible to enhance `impact` with other “connectors” to other existing package hosting solutions (private or public). As long as the schema is known to `impact` it can then pull its data from basically all possible places. For example, it would also be possible to use the API of the GitLab project[10] to extract the same information.

## 4 Package Index

Package information is maintained in an index file. This index file is also generated by `impact` but the process of building an index is not normally used by the user or tool vendors so all discussion about the creation of index files is presented later in Section 5. The index file is stored in JSON format and has the following structure:

```
{
  "<LibraryName>": {
    "homepage": "<URL>",
    "description": "<description string>",
    "versions": [
      "<version number>": {
        "version": "<version number>",
        "major": <major version number>,
        "minor": <minor version number>,

```

```

        "patch": <patch version number>,
        "tarball_url": "<URL to tarball>",
        "zipball_url": "<URL to zipball>",
        "path": "<path to library>",
        "dependencies": [
          {"name": "<DepLibName1>",
            "version": "<version string>"},
          {"name": "<DepLibName2>",
            "version": "<version string>"},
          ...
        ]
      }
      ...
    ]
  }
}
```

All quantities listed within angle brackets, `<...>`, are library specific details. The `<LibraryName>` is the name of the package in Modelica. Generally speaking, all version numbers follow the semantic versioning approach. However, since not all Modelica libraries currently follow semantic version conventions, indices can include semantic duplicates (*e.g.*, 1.0 and 1.0.0) which reference the same underlying version. Therefore, any non-semantic conforming versions (*e.g.*, 1.0) will act as “redirects” to the semantic version (*e.g.*, 1.0.0).

The `homepage` field is a URL to a web site that contains additional information about the library. The `zipball_url` and `tarball_url` fields point to archives that can be downloaded, in the zip and tar formats, respectively. The `dependencies` field lists all the library’s dependencies. These are the libraries that will also be installed when installing the specified library version they are listed under. The `path` field specifies the name of the directory or file representing the Modelica library within the specified archive.

Note, we have not currently defined a schema for this format. To promote interoperability we recognize that a formal schema would be the next logical step. We have added the creation of a JSON schema for the index file format to our list of next steps to promote interoperability with other implementations. Our hope is that such a schema would further encourage tool vendors to support this format as a means of publishing information about available Modelica libraries.

The Modelica Association index of publicly available libraries can be found at [https://impact.modelica.org/impact\\_data.json](https://impact.modelica.org/impact_data.json).

## 5 Private Packages

### 5.1 Using Private Indices

As mentioned in Section 4, there will be a package index hosted on `modelica.org` that lists any packages connected to special Modelica related GitHub accounts. This provides a means for library developers to quickly add their libraries to the set of libraries that are publicly indexed.

However, we recognize that many users will depend on libraries that cannot be hosted publicly. At the same time, we would like for those users to be able to benefit from the same kind of package management features for finding and installing their privately hosted libraries.

For this reason, users can create a special configuration file that lists the indices to be searched. By default, `impact` will use only the index hosted on `modelica.org`. But through custom configurations, users can specify any collection of indices (public or private) they wish to use.

To specify an alternative list of indices, a user would simply edit their user configuration file and add the following text:

```
[Impact]
indices=<url1>,<url2>
```

where the value of `indices` is a list of URLs pointing to index files. The default value for the `indices` variable is `https://impact.modelica.org/impact_data.json`. In cases where private index are files used, the URLs for private index files should be listed first and the URL to the index file hosted on `modelica.org` should be last.

Note, the location of the user's configuration file will depend on the platform they are using. Information about the location of the configuration file and current settings is generated by the following command:

```
$ impact info
```

### 5.2 Generating Private Indices

In order for users to include a private index file in the list of indices to be searched (as discussed 5.1), it is necessary to also have the capability to easily generate such private indices. This functionality is also available using the `impact` command line although we did not discuss it previously because it is not functionality that a typical user would require.

To generate an index file, the following `impact` command line syntax should be used:

```
$ impact refresh <source1> \
    <source2> ... <ourcen> \
    -o <output file>
```

where each source is a URL that encodes information about a potential source. For example, the default sources are `github://modelica-3rdparty/*` and `github://modelica/*` (in other words, all repositories belonging to the GitHub user `modelica-3rdparty` and all repositories belonging to the GitHub user `modelica`, respectively). Note that later sources have a higher priority than earlier sources. Also, at the moment the only types of repositories supported are GitHub repositories although by using a URL based approach it is easy to extend the possibilities to include Git, Subversion or other types of repositories as long as the information required for the index file is available.

The output file generated from this command should then be made accessible to users so they can incorporate it into the set of indices they search (see Section 5.1 for more details).

## 6 Source Code and Licensing

The `impact` project started off as a simple script, then a gist and eventually a complete repository. The repository for the source code is hosted on GitHub at `https://github.com/xogeny/impact`. Potential contributors are invited to fork the repository and add more functionality. Contributions to improve `impact` are very welcome.

The software is distributed under an MIT license. As such, there are no significant restrictions on using

the code in open-source, closed-source or commercial projects. In fact, we welcome vendor support and adoption. In addition to making the complete source code for the package manager available and documenting the functionality in this (freely downloadable) paper, we are also making the index data freely available from the `modelica.org` domain. We hope that all these measures will lead to the highest possible chance of adoption.

## 7 Conclusion

Inspired by the approach taken in Bower, we've created `impact`, a package manager for the Modelica eco-system. Like Bower, this approach is relatively light and relies heavily on the GitHub API to aggregate and index publicly available libraries. Also like Bower, our approach relies on semantic versioning, a widely adopted approach for associating concrete meaning to the various elements of a version. We use these meanings to help validate and organize the tags associated with Modelica libraries.

The `impact` tool also provides one of the key elements of a package manager, the ability to automatically pull in dependencies during installation. With this feature, users can list the libraries they directly depend on and `impact` will automatically install any additional dependencies. The dependency information is constructed automatically from the `version` annotation already present in Modelica libraries.

The index of publicly available libraries is hosted on `modelica.org`. But `impact` can also be used to index and install private libraries as well. All `impact` functionality (installing, searching, *etc.*) is available for both public and private libraries.

## References

- [1] easy\_install. *Easily download, build, install, upgrade, and uninstall Python packages.* 2014. URL: <https://pypi.python.org/pypi/setuptools>.
- [2] pip. *A tool for installing and managing Python packages.* 2014. URL: <http://www.pip-installer.org>.

- [3] npm. *Node Packaged Modules.* 2014. URL: <https://npmjs.org/>.
- [4] GitHub. *Build software better, together.* 2014. URL: <https://github.com/>.
- [5] Inc. Twitter. *Bower – A package manager for the web.* 2014. URL: <http://bower.io/>.
- [6] PyPI. *The Python Package Index.* 2014. URL: <https://pypi.python.org/pypi>.
- [7] Tom Preston-Werner. *Semantic Versioning 2.0.0.* 2014. URL: <http://semver.org/>.
- [8] GitHub Developers. *GitHub API v3.* 2014. URL: <http://developer.github.com/v3/>.
- [9] JSON. *JavaScript Object Notation.* 2014. URL: <http://www.json.org/>.
- [10] GitLab developers. *GitLab API.* 2014. URL: <https://github.com/gitlabhq/gitlabhq/blob/master/doc/api/README.md>.

# MoUnit — A Framework for Automatic Modelica Model Testing

Roland Samlaus<sup>1</sup> Mareike Strach<sup>1</sup> Claudio Hillmann<sup>1</sup> Peter Fritzson<sup>2</sup>

Fraunhofer IWES, Turbine Simulation, Software Development, and Aerodynamics<sup>1</sup>  
Department of Computer and Information Science Linköping University<sup>2</sup>

## Abstract

A vital part in development of physical models, i.e., mathematical models of physical system behavior, is testing whether the simulation results match the developer's expectations and physical laws. Creation and automatic execution of tests need to be easy to be accepted by the user. Currently, testing is mostly performed manually by regression testing and investigation of result plots. Furthermore, comparisons between different tools can be cumbersome due to different output formats. In this paper, the test framework MoUnit is introduced for automatic testing of Modelica models through unit testing. MoUnit allows comparison of Modelica simulation results with reference data, where both reference data and simulation results can originate from different simulation tools and/or Modelica compilers. The presented test framework MoUnit brings the widespread approach of unit testing from software development into practice also for physical modeling. The testing strategy that is used within the Modelica IDE OneModelica from which the requirements for MoUnit arose, is introduced using an example of linear water wave models. The implementation and features of MoUnit are described and its flexibility is exhibited through two test cases. It is outlined, how MoUnit is integrated into OneModelica and how the tests can be automated within continuous build environments.

*Keywords: MoUnit, OneModelica, Modelica, test framework, automatic testing, verification*

## 1 Introduction

An important part of the process of developing physical models is model testing since it greatly increases the probability that the expected behavior is accurately modeled. Within software development it is convenient to create tests and test suites using test frame-

works like JUnit<sup>1</sup>. The use of these test frameworks ultimately lead to a “test first” development approach, i.e., creating tests before actually developing the software parts that must conform to the tests. Also, engineers creating physical models can benefit from a test driven development. Immediate feedback about incorrect model changes can assist in keeping the models consistent during their lifecycle in the modeling process.

By using fine-grained tests for single components, errors in composed models can be traced back to the erroneous element. This also simplifies the development in teams as one modeler's changes may have a negative influence on another modeler's models. However, test-based development as known from software engineering is hard to maintain for physical model developed with Modelica. While the algorithmic parts of models could be checked in a similar way, the behavior expressed by equations cannot be tested directly since the usage of time as a parameter makes it necessary to perform simulations.

A reasonable solution for this is the use of regression tests where reference data is used as a base for comparison with results from subsequent simulations. When models are altered, the result can be compared to the reference files to check if the behaviour is still as expected. The two commonly used Modelica simulation tools OpenModelica<sup>2</sup> and Dymola<sup>3</sup> use regression tests for checking if changes made in the compilers still result in valid simulations as well as to allow users to create regression tests for model development.

Although these kind of tests already aid the development process of physical models, regression tests may not be sufficient to find all problems that may appear during the development and the tools provide a non-uniform way of defining test cases. With our test framework MoUnit we aim at facilitating test case and test suite definition for single model components as

<sup>1</sup><http://www.junit.org>

<sup>2</sup><https://www.openmodelica.org/>

<sup>3</sup><http://www.dymola.com>

well as groups of models. Besides, the tests should be executable with various simulation tools as well as in different environments, e.g., directly inside the Modelica IDE OneModelica<sup>4</sup>, or in continuous build environments like Hudson<sup>5</sup>, which is currently in use for OneModelica and OpenModelica development. Moreover, we want to allow users to extend MoUnit in order to re-use their already implemented test functionality as well as making it possible to create more sophisticated tests than regression tests.

In this paper, we demonstrate how the user can create test cases for automatic testing. The user can use arbitrary simulation tools, compare the results to reference data and to the results of a second experiment. Thereby, it is possible to compare simulation tools to each other or investigate the impact of different solvers on the simulation results without the need to create additional reference files. Moreover, we demonstrate the extensibility of the test framework for example through the implementation of a test component for regression test of equidistant result files. As a second tester component we implemented the calculation of basic statistics. This allows us to validate stochastic properties of our Modelica irregular water wave models by comparing to results from the commercial software ASAS WAVE<sup>6</sup>. Here it is advantageous that several input formats are supported in order to use data from tools that are not Modelica-based. Thus, calculations from analytic formulae producing numerical results can also be used for testing in our framework.

The remainder of this paper is structured as follows: In Section 2 we describe the project structure of Modelica projects in OneModelica. Separation into projects allows to separate code for testing from models containing the physical behavior. The test language is described in Section 3 along with an explanation of how the test framework can be extended. The evaluation in Section 4 demonstrates how the water wave models developed at Fraunhofer Institute for Wind Energy and Energy System Technology (Fraunhofer IWES) are tested using our framework. Finally, Section 5 concludes our work and gives an outlook on how we plan to further enhance the test framework in the future.

<sup>4</sup><http://www.onewind.de/OneModelica.html>

<sup>5</sup><http://hudson-ci.org/>

<sup>6</sup><http://www.ansys.com/Products/Other+Products/ANSYS+ASAS>

## 2 Testing Modelica Models Within OneModelica

Testing is a crucial part of model development. The following aspects of a model should be checked: whether it (a) can be simulated without errors, (b) delivers the expected results, and (c) represents the reality appropriately. Furthermore, when component models are developed and/or modified it also has to be checked whether the composed models deliver the expected results. That is, the robustness of a component model in various situations has to be ensured as — depending on the complexity of the model — even a small change might lead to unexpected results. Testing a model in different situations can be a tedious process as it involves a lot of “playing around with parameters” and waiting for simulations to finish. All these points are reasons for automated testing during model development.

The testing strategy which is presented herein and from which the requirements for the development of MoUnit arose, is closely tied to the Modelica IDE OneModelica. In OneModelica, models are organized in model projects [2]. This approach allows to have two model projects for each component model that has to be implemented. The first one contains all the physical system equations and model logic. However, the models are not yet simulatable because the parameters do not have values yet — as they should be modified by the user when the model is parameterized for various application scenarios. The second one contains the actual test models. The test models contain instances of the models taken from the original model project. The modifications of the original models are done with parameter settings that correspond to defined test case scenarios. This could also be a special set of parameters that has caused problems in the past and it should be ensured for future simulations that it will not occur again. This strategy is somewhat borrowed from the software development: JUnit test plug-ins will be separated from the actual Java plug-ins to divide program logic from the test. The use of instances of the models to be tested rather than extending it, is based on the fact that test models can also be seen as “best practice” examples: a look at the test models shows how the model to be tested should be used. However, if this aspect is not important, it is also possible to use a test model that extends the model to be tested within MoUnit. This would avoid possible inconsistencies between the physics and test model code that is duplicated such as parameter units. Using the ex-

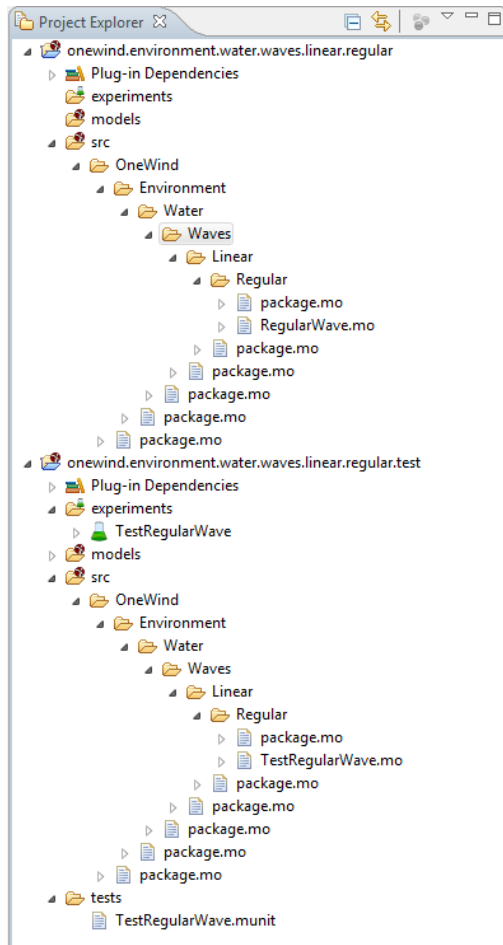


Figure 1: Model projects for both model logic and test models for regular water waves

ample of Modelica models for regular water waves, this is illustrated in Figure 1: The test model project has the same name as the original model project, but with an appended `.test`. This makes it clearly distinct as a test project for all the developers, who work on the models. As can also be seen in Figure 1, the test model project contains experiments that simulate the test models. The structure of the model to be tested and the testing model is shown in Listing 1: the model `RegularWave` has the parameter instances and equations, while the model `TestRegularWave` has an instance of type `RegularWave` and modifies the corresponding parameters. The experiments that are used to simulate the test models are set up manually. They can also be simulated manually during the model development. However, once the number of models and functions has exceeded some threshold been created, manual execution of all the experiments becomes too cumbersome. This is the point where MoUnit should be used. The necessary files for automated execution

```

model RegularWave
  "kinematics of a regular linear wave"
import SI = Modelica.SIunits;
import NonSI =
  Modelica.SIunits.Conversions.NonSIunits;

parameter SI.Height H "wave height";
parameter SI.Period T "wave period";
parameter NonSI.Angle_deg epsilon "phase";
// rest of parameters omitted

equation
  // model logic is placed here
end RegularWave;

model TestRegularWave
  "test model for 'RegularWave'"
import SI = Modelica.SIunits;
import NonSI =
  Modelica.SIunits.Conversions.NonSIunits;

constant SI.Height H = 5;
constant SI.Period T = 11;
constant NonSI.Angle_deg epsilon = 12;

RegularWave regularWave(H = H, T = T,
  epsilon = epsilon
  // rest of modifications omitted
) "instance to be tested";

equation
  // test logic like assert statements
  // is placed here
end TestRegularWave;

```

Listing 1: Example for model and corresponding test model

of the experiments are stored in the test model project as shown in Figure 1. These files enable checking that all the models that have been implemented in the original model project are fulfilling the three properties mentioned above: it can be easily seen if a simulation crashes and if the implementation of the model works correctly by comparing it to reference data that can either come from other simulation tools, measurements, or analytical solutions.

### 3 Implementation of the Modelica Test Framework

For the development of valid physical models with Modelica it is vital to define test cases to check that model changes do not negatively affect simulation results. A common way to assure this is using regression tests. This is used by the OpenModelica development group using their framework to test compiler changes by comparing results of simulations to reference data. In the same way Dymola's model management library allows performing regression tests with

Modelica models.

The goal of MoUnit is to provide test-driven physical model development with Modelica. For this purpose we implemented a test framework that is integrated into the Modelica IDE OneModelica and is flexible to be also used in automatic build environments like Hudson. A special test language has been defined that can be parameterized with custom test components and test result listeners which are responsible for processing test results appropriately.

In the following, the test language is described that has been developed with Xtext [3]. It allows defining test cases and test suites for automatic execution. Additionally, the extension mechanism is explained to show how custom test components can be registered allowing users to write test cases for special purposes — for example, basic statistics analysis. Subsequently, the test runner that performs the test execution is described and the registration of listeners is demonstrated. The listener mechanism allows to use MoUnit in different environments, e.g., by displaying the test results visually inside OneModelica or by generating text files for the integration of tests in continuous build environments like Hudson. We have implemented an ANT task that is executed by Hudson. The task adds a listener to the test runner creating a XML file for the documentation of the test results. The result file can then be used by Hudson for error reports.

### 3.1 Test Definition Language

Xtext is a tool for textual language development. By providing a grammar definition of a language, editors and views can automatically be generated with helpful functionality like syntax highlighting and syntax checking. The Modelica editor being the basis for OneModelica has been developed with Xtext. Parsed documents are represented as Ecore-based [4] parse trees and can be used along with other Ecore-based languages (e.g., cross-references between elements of different languages are supported). Since mandatory components for automatic test execution in our IDE are defined as Ecore-models, those components can easily be re-used for MoUnit.

Two components can be defined with the test language that are comparable to concepts used by the test framework JUnit for Java code: Test cases and test suites. Test cases (see Listing 2) define the basic elements that are needed for a test to run while test suites (see Listing 3) combine test cases and other test suites. Hence, a set of test cases can be accumulated to test the correct behavior of interacting Modelica models.

```
name TestCase1
experiment ToTest
file "pathToFile.mat"
compare {
  qualifiedname.*
  test.qn -> reference.qn

  TestComponent {
    attribute = 1.e-5
  }
  error "Error message"
}
```

Listing 2: Example of a Test Case Definition

```
suite AllTests:

  TestSuite1,
  TestCase1
```

Listing 3: Example of a Test Suite Definition

Test cases define names that make them referenceable and thus usable in test suites. The `experiment` keyword states that a Modelica experiment is used in the test. Modelica experiments in OneModelica can use several tools for simulation, including OpenModelica and Dymola. For regression tests a path to a reference file is provided following the keyword `file`. Several checks were implemented to display problems to the user, e.g., when the provided file cannot be found or the experiment is set to leave the Dymola window open after simulation for post processing. This would cause the test execution to pause until the user closes the window manually. It is also possible to directly compare the test results of two experiments by using a second experiment instead of the reference file. This can help to ensure that the results of simulations with OpenModelica are the same as those obtained from simulations with Dymola. Another possibility is to check if the results are equal for different kinds of solvers.

It is also possible to compare two result files with each other, e.g., when it is desired to compare results from other modeling tools and languages. The result files can be in `.mat` format as provided by OpenModelica and Dymola (textual or binary format) as well as in comma separated values format. Moreover, the user can also create results based on some analytical formulae, if it is not desired to implement the analytical result as a test component to register it to the test framework, and save them in one of the supported formats instead.

Simulation result files can be referenced by abso-



lute or relative paths. Using relative paths, files can be put into a project inside OneModelica's workspace and can be synchronized by multiple users with source code management tools like SVN. Since simulation result files might be large, it may be necessary to use network drives for storage. This can be done by using absolute paths that can be prepended with the prefix \$. The prefix will be replaced by a user defined path. This way tests are re-usable for other users, and the prefix only has to be configured once per workspace.

Now that the data for comparison is available, the user needs to provide the qualified names of model elements that shall be compared, following the compare keyword. If no element is defined then all time series from the results are compared to each other. The user can use the asterisk character \* to select groups of elements.

Next, test components are referenced that perform the comparison of results. The components are registered via an Eclipse extension point [5, 6]. The test components must implement a predefined interface and need to be modeled with Ecore. Attributes can be defined which must be parameterized by the user. In the provided example a tolerance for the result comparison is defined which will allow a small difference in the result data of the compared time series.

Possible values for the parameterization are Doubles, Integers and arrays of Doubles and Integers. The attributes are looked up by introspection [9] which is used to validate whether the user provides correct input. When the validation is performed a new instance of the test component is created and the parameters are set. Finally the user defines an error message that will be displayed when a test run fails.

Test suites start with the keyword `suite` followed by a name. They contain a list of test cases and additional test suites. All test cases contained in a test suite will be performed when the suite is executed.

### 3.2 Test Execution and Result Processing

The test definitions are processed by a Modelica test runner. The runner is implemented without dependencies to user interface components to allow usage in build environments that do not have graphical user interface installed (e.g., Linux operating systems on dedicated build servers). The test runner executes the experiments used in the test cases and compares the results to the referenced experiments or files. Listeners can be registered to the test runner to be informed about the executed tests and test results. The duration of each test is measured for performance analysis. The

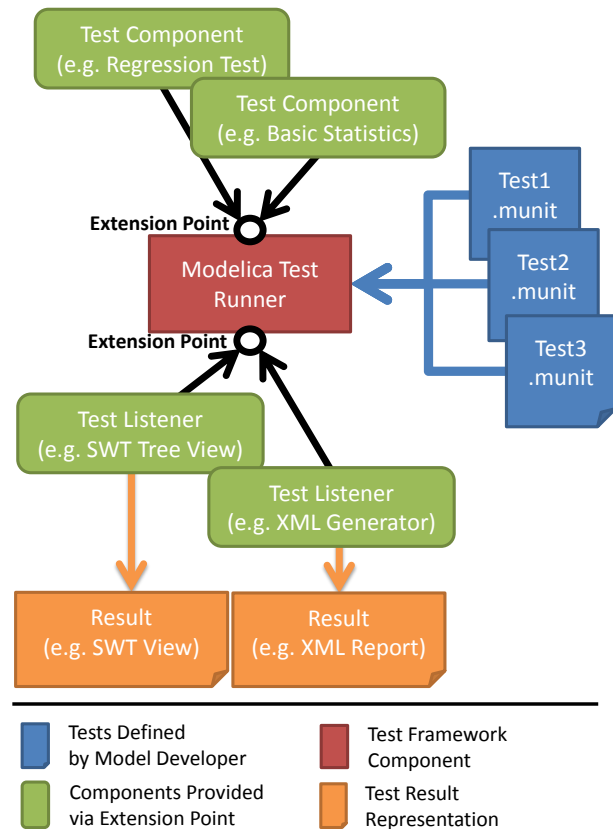


Figure 2: Structure of the Modelica Test Framework

test components must return a status object that provides information about whether the test succeeded or failed. When a test failed, a message must be provided and an optional index (e.g., the time step) can be used to tell the user about the erroneous location in the result file.

Besides the basic information described above, a test component can return arbitrary objects with additional information about an error. This can, for example, be an SWT<sup>7</sup> composite object for visualization inside OneModelica or a file path to a serialized plot which can be used for a test report. The test runner listener is responsible for checking whether it can handle the object or not.

For the visualization of test results in OneModelica we implemented a view (see Section 4) that is comparable to JUnit's test view. Test suites and test cases are displayed as a tree and the nodes are marked according to successful or erroneous execution. While JUnit test cases are based on assert statements that check whether particular parts of code run correctly, we need to compare result files to check whether the desired behavior is matched. This results in a different kind

<sup>7</sup><http://www.eclipse.org/swt/>

of test status visualization where the qualified names of erroneous models is provided and optional graphs can visualize deviations between expected and present values. A JUnit like behavior could be implemented for algorithmic code, e.g. by interpreting the algorithms [1], allowing to check code very fast. However, this is currently not yet supported by MoUnit.

For the integration with automatic build environments we developed an Eclipse ANT task<sup>8</sup>. The task is performed on an Eclipse workspace that contains the Modelica projects as described in Section 2 with experiments and test definitions. The user provides information about which test cases and test suites shall be executed. For test logging we implemented a test runner listener that creates XML files according to the format used by the JUnit ANT task. Hence, failing Modelica tests can be handled in the same way as JUnit tests, e.g. by sending out notification mails to the responsible model developers. The ANT task enables the use of the same test definitions in continuous build environments as well as for manually triggered testing inside OneModelica.

## 4 Evaluation

In the following, the example of linear water wave models is used for the evaluation of MoUnit. These models were developed at Fraunhofer IWES to display both regular and irregular waves according to Airy wave theory with corresponding stretching methods. The underlying theory for these models can be found in standard offshore engineering text books like [7] or [8].

For evaluation purposes, the variable *eta* corresponding to the wave elevation, i.e. the instantaneous wave height at a given position, is used. For a regular linear wave, *eta* is deterministic as it can be displayed by a cosine function. That is, it can be compared to reference data through a regression test. Accordingly, this is used as a first test case denoted as `TestRegularWave`.

The second test case is called `TestIrregularWave` and tests the variable *eta* for an irregular linear wave. In this case, *eta* is stochastic, i.e., a direct comparison of time series to reference data is meaningless unless they are generated with the exact same input. However, this is not always possible, e.g. when the simulated time series are compared to reference data from a different tool or measurements. Instead, the mean

```
name TestRegularWave
experiment
TestRegularWheelerWave
file
"$/WAVE_WheelerElevation.csv"
compare {
    regularWaveKinematics [1].eta
    -> asas.elevation.eta

    EquidistantTSValidator {
        tolerance=10e-11
    }
    error "Wave elevation does not match."
}
```

Listing 4: Test Definition for Regular Waves

value and the standard deviation of the time series are compared to those of reference data.

For the test case `TestRegularWave` (see Listing 4), we implemented the test component `EquidistantTSValidator`. It can be parameterized with a `tolerance` that represents the acceptable absolute difference between the two compared time series. The input files for the component must have the same length as well as equidistant values. If the input does not match the reference data, i.e., if the absolute difference between input and reference data exceeds the value of `tolerance`, an error will be displayed. The resulting view for our example is shown in Figure 3: the simulation time plus the time used for validating the values is displayed as a sum in the test case tree view. When the user selects the erroneous test, basic information about the location of the first assert failure is provided together with a plot of the two compared curves.

For the test case `TestIrregularWave` (see Listing 5) a test component named `BasicStatistics` has been implemented to compare mean value and standard deviation as described above. The user can provide the parameters `startindex` and `endindex` to define the range of values that shall be checked for the provided input values.

```
name TestIrregularWave
file
"$/ir_periodical_elevation_dymola.csv"
file
"$/ir_elevation_asas.csv"
compare {
    modelica.elevation.eta
    -> asas.elevation.eta
```

<sup>8</sup><http://ant.apache.org/>

```

BasicStatistics {
    startindex=0,
    endindex=800,
    meandeviation=0.01,
    stddeviation=0.001
}
error "Basic statistics of wave
    elevation do not match."
}

```

Listing 5: Test Definition for Irregular Waves

Additionally, the user can define the allowed variance of the calculated mean values and standard deviations. Figure 4 shows the resulting view displaying information about the assertions that were violated. While the mean value of the wave elevation matches the reference data, the standard deviation is not acceptable. As for the test case `TestRegularWave`, a plot is provided to visualize the compared values to the user.

## 5 Conclusion and Outlook

In this paper, we demonstrated that the test framework MoUnit developed at Fraunhofer IWES can effectively be used for testing Modelica models by unit testing. We showed that the framework is flexible as it supports multiple simulation tools as well as input files from other tools for result comparison. This allows to check the different behavior of models when using different simulation tools as well as different solvers.

We also illustrated that it is possible to extend MoUnit with custom test components that provide functionality different from regression testing. Using the example of simulating water waves, we compared simulation results of our Modelica models to simulation results of the reference tool ASAS WAVE. The extensibility also allows to add test components that use results from analytical calculations for result validation.

For error reporting, a flexible interface has been defined. We implemented a plot for wave model tests displaying the time series of the data that is being compared together with the corresponding mean values and standard deviations. Thereby, the user can directly investigate the cause for a problem visually. Furthermore, an ANT task has been implemented that executes test suites and registers a listener that serializes a test report in an XML file compatible to JUnit. Hence, the tests can be integrated into a build system for continuous testing.

In the future, we plan to create additional test functionality that allows to test single values of result files. The implementation of test components for analytical

solutions are planned for the models of wind turbine components — as far as they are available. For fast testing of algorithms, it is furthermore desirable and planned to interpret the algorithmic code, e.g. of Modelica functions to allow direct calls and comparison to expected result values.

## Acknowledgements

This work is financially supported by the Federal Ministry for the Environment, Nature Conservation and Nuclear Safety based on a decision of the Parliament of the Federal Republic of Germany, by the Swedish Government in the ELLIIT project, by the Swedish Strategic Research Foundation in the EDOP projects, and by Vinnova in the RTSIM and ITEA2 MODRIO projects.

## References

- [1] Höger C. Modelica on the Java Virtual Machine. In: Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, April 19 2013, University of Nottingham, Nottingham, UK.
- [2] Samlaus R, Hillmann C, Demuth B, Krebs M. Towards a Model Driven Modelica IDE. In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany, Modelica Association, 20-22 March 2011.
- [3] Köhnlein J., Efftinge S., Xtext 2.1 Documentation, Itemis GmbH, October 31, 2011.
- [4] Budinsky F, Brodsky S. A., Merks E. Eclipse Modeling Framework, Pearson Education, 2003.
- [5] Clayberg E, Rubel D. Eclipse Plug-ins, Addison Wesley Professional, 2009.
- [6] McAffer J, van der Lei P, Archer S. OSGi and Equinox: Creating Highly Modular Java Systems, Addison-Wesley Professional, 2010.
- [7] Faltinsen O M. Sea loads on ships and offshore structures, Cambridge University Press, 1990.
- [8] Chakrabarti S K. Ocean Environment. In: Chakrabarti S K (ed.), Handbook of Offshore Engineering, Elsevier, 2005.
- [9] Eckel B., Thinking in Java, Prentice Hall, 978-0-13-187248-6, 2006.

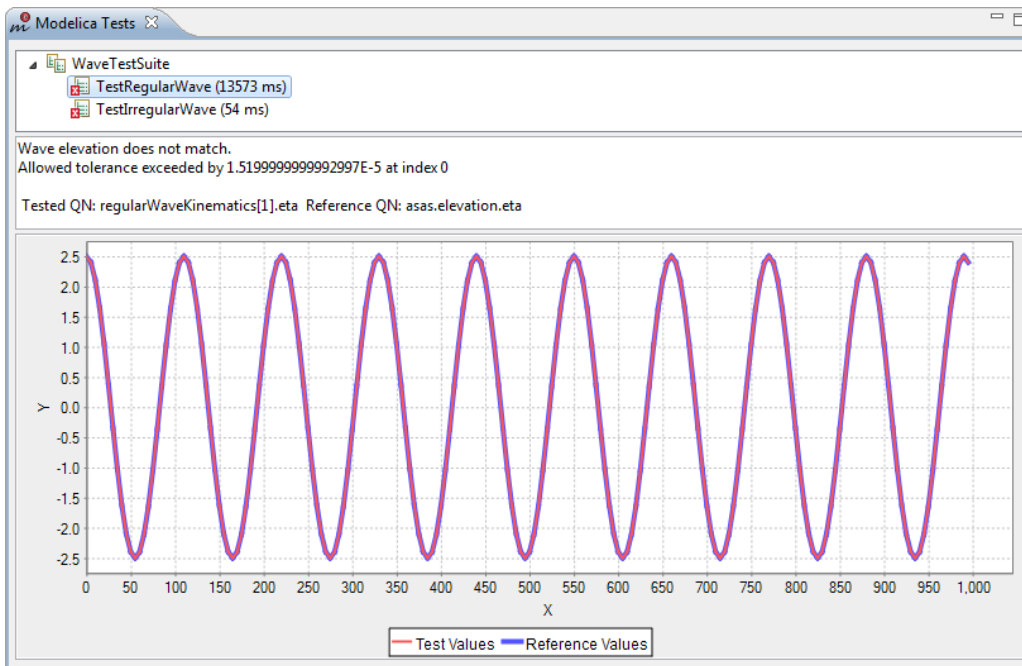


Figure 3: Result for test case TestRegularWave

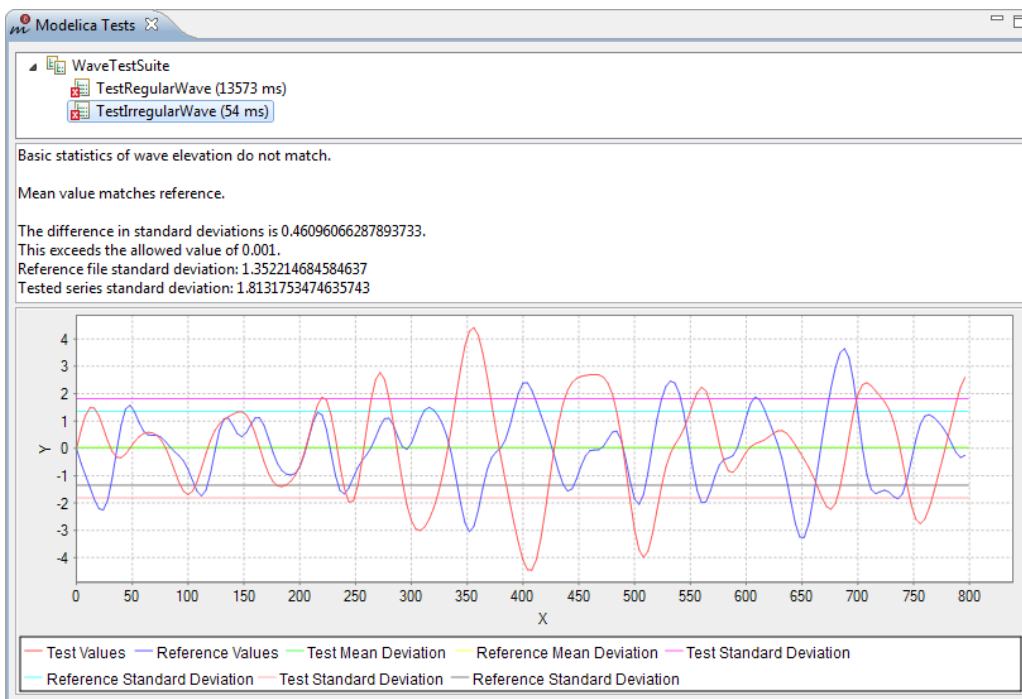


Figure 4: Result for test case TestIrregularWave

# Modeling Parameter Sensitivities via Equation-based Algorithmic Differentiation Techniques: The ADMSL.Electrical.Analog Library

Atiyah Elsheikh  
Austrian Institute of Technology, Vienna, Austria  
Atiyah.Elsheikh@ait.ac.at

## Abstract

Parameter sensitivities of mathematical models play a vital rule in many applications of sensitivity analysis. The availability of algorithmic capabilities for representing and computing these quantities is surly advantageous. In this work it is shown how to systematically transform a Modelica library to another library that describes the desired models together with derivatives of model variables w.r.t. model parameters. The produced library remains with the same structure and the underlying models keep the same interface and outlook. The proposed approach relies on novel equation-based algorithmic differentiation techniques that are especially designed for Modelica. The illustration is rather done via a compact library, the open-source ADMSL library, but rich enough to facilitate a lot of representative Modelica language constructs. The ADMSL library is the algorithmically differentiated version of the standard Modelica library subpackage `Modelica.Electrical.Analog.Basic`.

*Keywords: algorithmic differentiation; parameter sensitivities; sensitivity analysis, ADMSL, AD of Modelica libraries*

## 1 Motivation to algorithmic differentiation

If you want to draw Bamboos, you should try drawing Bamboos for your whole life, then you might be able to draw Bamboos, an ancient Chinese wisdom. In other words, the earliest modelers have already recognized that there is no model that identically describes the reality up to the tiniest details. This seems to be also the case now days, at least in the field of Systems Biology [27]. However even an elementary model describing a complex system is an essential initial step towards gaining insights and

winning additional knowledge of the modeled system. Via e.g. the availability of further experimental data, the model can be better tuned [5]. Nevertheless, instead of trying to approach a true model for one's whole life, many tools of sensitivity analysis, such as model identification, validation and optimization [11, 12] can help the modelers to realize their visions, hopefully in a reasonable amount of time.

Significant quantities, that can assist the implementation of such computational tools, are parameter sensitivities, i.e. the derivatives of model outputs w.r.t. model parameters. Straightforward ways for evaluating these quantities via finite difference methods are not recommended for accuracy reasons [16]. A more reliable but technically difficult approach is to symbolically derive these quantities and then to evaluate them via a numerical integrator, e.g. the IDAS solver within the Sundials Suite [21].

These factors among other typical applications of mathematical derivatives increasingly attract attention at a special domain of scientific computing called Algorithmic Differentiation (AD) of computer programs [19]. Based on the chain rule of Calculus, procedural compiler methods and other established techniques, many automatic differentiation tools (cf. [www.autodiff.org](http://www.autodiff.org)) are capable of:

- analyzing a large set of computer programs written in many procedural programming languages
- adequately computing new programs additionally representing partial derivatives

The resulting generated programs are typically used for evaluating the derivatives of program outputs w.r.t. program inputs among other directional derivatives.

## 2 Introduction

### AD and the Modelica community

While classical AD techniques and tools are targeting a large scope of developers of mathematical programs, only a tiny part of the Modelica community is explicitly utilizing AD methods, namely tool vendors and developers of Modelica simulation environments. For instance, in [25, 4, 7] the Jacobian is symbolically computed for the task of index reduction and adaptive numerical integration. For the more difficult task of computing parameter sensitivities, some simulation environments such as JModelica [3] and SystemModeler [2] can be used. A translated Modelica model is linked with advanced specialized integrators suite like SUNDIALS capable of evaluating parameter sensitivities [21]. In this context, AD techniques are usually applied at low level C code describing the translated equations.

In contrary, the approach of applying AD techniques directly to the high-level descriptive model represents another attractive option. An example is given by the tool ADModelica [15, 14], which is capable of transforming a given high-level Modelica model into another Modelica model additionally describing parameter sensitivities. Similar works have been also reported with other modeling languages [6, 20]. Nevertheless and even with the availability of open-source developer-oriented compiler tools like OpenModelica [26], the development and maintenance efforts for such a tool attempting to cope with a rich language like Modelica becomes a continuously exhaustive task.

### Contribution

For the first time an equation-based modeling-oriented AD approach for differentiating Modelica libraries is demonstrated. The approach provides the basic guidelines for systematically transforming a library into a topologically-identical algorithmically differentiated library in which parameter sensitivities are additionally represented. By reimporting the augmented library into already existing base models and slightly altering the declaration for specifying the required model parameters w.r.t. which derivatives are sought, parameter sensitivities is represented at the model level. In this sense, transformed library components are overloaded with semantics for describing parameter sensitivities. Using an arbitrary Modelica

simulation environments, parameter sensitivities are directly simulated within the model.

Due to the speciality of the Modelica language and its significant difference from assignment-based procedural languages for which classical AD techniques were designed, new specialized equation-based AD techniques are demonstrated in this work. These techniques, relying on equation-based compiler notions, utilize Modelica powerful capabilities to provide efficient representation of partial derivatives. Consequently, the inclusion of parameter sensitivities within Modelica libraries can be considered from the early design phase or alternatively, existing libraries can be augmented with additional components for describing the required derivatives. In this sense, AD techniques needs to be applied only once and the resulting library can be used for ever.

According to Naumann [24], to master AD concepts, AD users should be first able to manually differentiate their own compact programs before applying AD tools. In this work, the presented techniques are intuitive enough to be systematically applied on manual basis even on sophisticated libraries with complex mathematical models, e.g. ADGenKinetics [10]. In this way, a larger part of the Modelica community can possess the art of self-handing AD techniques along their modeling activities. The presented approach describes the basic steps subject to automation and consideration within realistic Modelica-based AD tools. A comprehensive algorithmic specification of the demonstrated techniques is given in [13].

## 3 General scheme

The key idea of the presented approach is illustrated via the diagram presented in Figure 1. Without loss of generality, a model  $M$  with two connected components  $C_1$  and  $C_2$  is given. The components are mathematically described by Differential Algebraic Equations (DAEs) via the functions  $f_1$  and  $f_2$ , respectively.  $x_i$  and  $p_i$  for  $i \in \{1, 2\}$  correspond to systems variables and model parameters, respectively. The function  $g$  describes a causal or an acausal connection relation between  $C_1$  and  $C_2$ .

The algorithmically differentiated components  $C'_1$  and  $C'_2$  extend the components  $C_1$  and  $C_2$  with the underlying Sensitivity Equation Systems (SESs). A SES is obtained by *forward differentiation* of

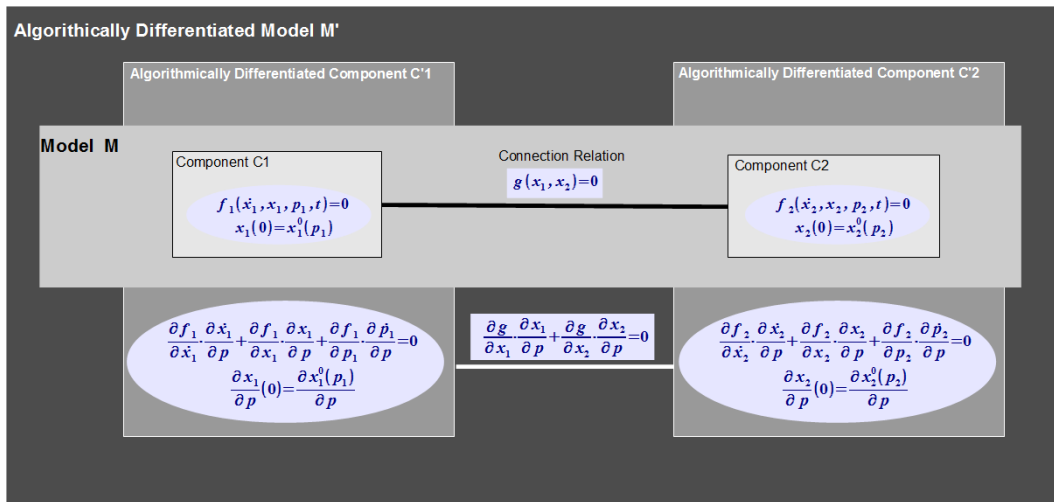


Figure 1: A general block diagram of a model with two connected components and their algorithmically differentiated copies

the original DAE system w.r.t. the model parameters  $p = (p_1 \ p_2)^T$ . The component  $C'_1$  additionally declares the *input Jacobian* of the parameters  $\partial p_1 / \partial p$ , i.e. the derivatives of the local *active parameters* within  $C_1$  w.r.t. the parameters of the whole model  $p$ . In general, the model parameters are of course not known within the separate component  $C'_1$ . Therefore, input Jacobian  $J_p$  is declared w.r.t. arbitrary model parameters  $p$  that are first defined at the model level. An example is given in the next Section. Analogously  $C'_2$  specifies the active local parameters within  $C_2$  via their input Jacobian w.r.t. arbitrary unknown parameters.

An algorithmically differentiated model  $M'$  is obtained by connecting the components  $C'_1$  and  $C'_2$ . The connection relation is similarly described by forward differentiation of the equation system  $g$  w.r.t.  $p$ . The key idea is based on the chain rule of Calculus by which partial derivatives are propagated among components and hence parameter sensitivities of the whole model are inherently present.

There are many structural similarities between the model  $M$  and its AD version  $M'$  [17]:

1. Both models have the same interface in a GUI-editor, the SES remains hidden from the user perspective
2. The Jacobian of the differentiated equation system w.r.t. one single parameter has an identical structure and sparsity pattern of the Jacobian of the original system [23]

3. It can be proven that the structural indices of both models are equal

These characteristics can be utilized for simplifying the compilation phases within a Modelica compiler targeting AD of Modelica libraries and models. The next section provides a simple example illustrating the whole paradigm in Figure 1.

## 4 Illustrative example: The declaration part

### 4.1 The ADMSL library

Based on the techniques presented, an example illustrated on a subpackage of the Modelica standard library (MSL) `Modelica.Electrical.Analog.Basic`. The whole illustration is provided via the open-source ADMSL library [1]. The ADMSL library serves as an experimentation platform for:

1. Illustrating the basic steps for performing AD of Modelica libraries
2. Identifying the best capabilities of the Modelica language relevant for expressing parameter sensitivities at the library components level
3. Recognizing current limitations from expressibility perspective with which automatic code generation becomes less systematic

## 4.2 Code generation rules

In Naumann [24], basic code generation rules for performing AD of procedural languages have been presented. Analogously, in this paper simple code generation rules, however, for the Modelica language are demonstrated. These Modelica-specialized rules serve as main guidelines for performing AD of Modelica libraries. Note that the presented rules are distinguishably different than those provided for classical languages. For a complete understanding of the following subsections, the reader is recommended to consult the subpackage `Modelica.Electrical.Analog`.

### Declaration rule 1: structure duplication

When performing AD of a Modelica library, all packages and subpackages are simply duplicated. For instance, the subpackage `ADMSL.Electrical.Analog` corresponds to the subpackage `Modelica.Electrical.Analog`. Within each package, model components with the same names are placed. Each model component extends the original components. For instance, the corresponding AD version of the connector `Pin` becomes:

Listing 1: AD version of `Pin`

```
connector Pin
  extends Modelica.Electrical.Analog.
    Interfaces.Pin;
  ...
end Pin;
```

For components declaring these Pins, the corresponding AD version, e.g. of the `OnePort` component, should declare the corresponding AD versions of Pins. This is done by redeclaring replaceable Pins:

Listing 2: AD version of `OnePort`

```
partial model OnePort
  extends
    Modelica.Electrical.Analog.
    Interfaces.OnePort(
      redeclare ADMSL.Electrical.Analog.
        Interfaces.PositivePin p,
      redeclare ADMSL.Electrical.Analog.
        Interfaces.NegativePin n);
  ...
end OnePort;
```

The above code assumes that the electrical pins `p` and `n` are declared as `replaceable`. However, this is not the case in the latest Modelica library version 3.2. Alternatively, as a temporary solution, the original `TwoPort` model has been slightly modified and placed under `Analog.Interfaces.Bases` package as shown in Appendix A. Similarly, for models (e.g.

`Capacitor`) extending partial models (e.g. `OnePort`), their AD versions need to extend the AD versions of these partial models, for example:

Listing 3: AD version of `Capacitor`

```
model Capacitor
  extends Bases.Capacitor(
    redeclare replaceable class
      OnePort = ADMSL.Electrical.Analog.
        Interfaces.OnePort);
  ...
end Capacitor;
```

### Declaration rule 2: duplication of data segments

Any new component needs to reference the global number of active parameters w.r.t. which derivatives are sought. This is done by extending the component `ADMSL.Interfaces.GradientInfo` declaring global gradient related information:

Listing 4: Declaring the number of gradients

```
outer parameter Integer NG
  "dimension of gradient";
```

The parameter is declared as `outer` utilizing implicit connection mechanisms. It gets first initialized only by explicit declaration defining the same parameter as `inner` at the top level model. Accordingly, for all model components a *derivative object* (i.e. the gradient) is additionally declared for each real variable or parameter within the corresponding component, e.g. the AD version of the connector `Pin`:

Listing 5: AD version of `Pin`

```
connector Pin
  extends ... // as before
  extends ADMSL.Interfaces.GradientInfo;
  Real g_v[NG]
    "gradient of the voltage";
  flow Real g_i[NG]
    "gradient of the current";
end Pin;
```

The idea is simple, potential derivative objects are associated with variables while flow derivative objects are associated with flow variables. Similarly, in the model capacitor:

Listing 6: Declaration part of `Capacitor`

```
model Capacitor
  extends ...; // as before
  parameter Real g_C[NG] = zeros(NG);
  ...
end Capacitor;
```

Here, a derivative object initialized to the zero vector is associated with the parameter `C`. Each entry of



a gradient represents the derivative w.r.t. a parameter specified via the input Jacobian at the top model. So far the main rules for altering the declaration part has been demonstrated. In the next section, further rules for deriving the sensitivity equations are illustrated.

## 5 Illustrative example: The equation part

### 5.1 Modeling SES in gradient format

Assume that a given Modelica component  $C$  is described more or less DAE system (without discrete variables):

$$F(\dot{x}, x, p, t) = 0 \quad , \quad x(t_0) = x_0(p) \quad (1)$$

where  $x(t) \in R^n$  and  $p \in R^m$  represent state variables and parameters, respectively. Additionally, assume that  $F : R^{2n+m+1} \rightarrow R^n$  is continuously differentiable w.r.t.  $\dot{x}, x$  and  $p$ . Required is an additional component  $C'$  that augments  $C$  with additional equations for describing the time-dependent parameter sensitivities  $(\partial x / \partial p)(t) \in R^{n \times m}$ . As mentioned before,  $C'$  needs to include the *sensitivity equation subsystems* (SESub):

$$F_x \dot{s}_i + F_x s_i + F_{p_i} = 0 \quad , \quad s_i(t_0) = \frac{\partial x_0(p)}{\partial p_i} \quad (2)$$

$$\text{where } s_i = \frac{\partial x}{\partial p_i} \quad \text{for } i = 1, 2, \dots, m$$

SESub is obtained by differentiating all equations w.r.t. desired parameters. This is a large equation system of dimension  $m \cdot n$ . Explicit listing all these equations makes  $C'$  not compactly implemented.

To overcome this drawback, Modelica array capabilities can be utilized if the equation system 2 is implemented in gradient format. Within a model component corresponding to a DAE of the form (1), assuming that  $F_i$  corresponds to the  $i$ -th equation in  $F$  let  $F_x$  be defined as follows:

$$F_x = \left[ \frac{\partial F_i}{\partial x_j} : i, j = 1, 2, \dots, n \right] \in R^{n \times n}$$

and let  $F_{\dot{x}}, F_p, \dot{x}_p$  and  $x_p$  be analogously defined. Furthermore, let the input Jacobian  $J_p$  be defined as follows:

$$J_p = \left[ \frac{\partial p_i}{\partial p_j} : i, j = 1, 2, \dots, m \right] \in R^{m \times m}$$

A typical case is to set  $J_p = I_m$ , the identity matrix, for a set of independent parameters. Then the corresponding differentiated equation w.r.t. a parameter  $p_j$  is derived from Equation (2) as follows:

$$\sum_{l=1}^m \left[ \sum_{k=1}^n [F_{\dot{x}}(i, k) \dot{x}_p(k, l)] + \sum_{k=1}^n [F_x(i, k) x_p(k, l)] + \sum_{k=1}^m [F_p(i, k) J_p(k, l)] \right] J_p(l, j) = 0 \quad (3)$$

Using Modelica array capabilities and assuming that the given set of parameters is independent (i.e.  $\partial p_i / \partial p_j = 0$  for  $i \neq j$ ), Equation (3) can be rewritten in a gradient format comprising  $m$  equations:

$$\sum_{k=1}^n [F_{\dot{x}}(i, k) \dot{x}_p(k, :)] + \sum_{k=1}^n [F_x(i, k) x_p(k, :)] + F_p(i, :) = 0 \quad (4)$$

for  $i = 1, 2, \dots, n$  where  $A(i, :)$  represents the  $i$ -th row of a matrix  $A$ .

### 5.2 Code generation rules

#### Forward differentiation rule 3: deriving sensitivity equations

Using Equation (4), SES of simple model components can be easily implemented. For instance the equation part of the component `ADMSL.Analog.Basic.Capacitor` becomes:

Listing 7: Equation part of Capacitor

```
model Capacitor
  extends ...;
  ...
  equation
    g_i[1:NG] = g_C[1:NG] * der(v) +
                C * der(g_v[1:NG]);
  end Capacitor;
```

Deriving SES for simple mathematical formulas like the previous one is straightforward. For long complex formulas, common computer algebra packages can be used. However, it is recommendable to employ the equation-based AD techniques illustrated in Section 6.

#### Forward differentiation rule 4: handling blocks within flow control

Modelica, as many other languages, provides classical language constructs for flow control such as `for`, `while`, `if`,...etc. In classical AD concepts, assignment blocks within typical control flow constructs

(e.g. for, if, ... etc.) are directly differentiated as an independent block. Although such constructs are interpreted differently in an equation-based context, the generation of SES within such blocks are similarly intuitive. For example, the equation part of the model Analog.Examples.Utilities.NonlinearResistor is implemented as follows:

Listing 8: Equation part of NonlinearResistor

```

model NonlinearResistor
...
equation
  i =
    if (v < -Ve) then
      Gb*(v + Ve) - Ga*Ve
    else if (v > Ve) then
      Gb*(v - Ve) + Ga*Ve
    else
      Ga*v;
end NonlinearResistor;
    
```

The implementation of its AD version becomes:

Listing 9: Equation part of NonlinearResistor

```

model NonlinearResistor
...
equation
  g_i[:] = if (v < -Ve) then
    g_Gb[:] * (v + Ve) +
    Gb * (g_v[:] + g_Ve[:]);
  else if (v > Ve) then
    ...
  else
    ...;
end NonlinearResistor;
    
```

## 6 Equation-based AD

So far, the demonstrated components have short equations that can be easily differentiated. In this section, an equation-based AD technique, especially designed for the Modelica language, is shown. In the first ever algorithmically differentiated library, ADGenKinetics, the following equation corresponding to the convenience kinetics of chemical reaction rates

$$v = \prod_a \frac{K_{A_a} + [A_a]}{K_{A_a}} \cdot \prod_b \frac{K_{I_b}}{K_{I_b} + [I_b]} \cdot \frac{V_{max}^{fwd} \prod_i \frac{[S_i]}{K_{mS_i}} - V_{max}^{bwd} \prod_j \frac{[P_j]}{K_{mP_j}}}{\prod_i \left(1 + \frac{[S_i]}{K_{mS_i}}\right) + \prod_j \left(1 + \frac{[P_j]}{K_{mP_j}}\right) - 1} \quad (5)$$

has been easily differentiated using the demonstrated technique, consult [10] for more details about this

equation and the ADGenKinetics library. In this paper, the technique is illustrated on a more simple equation within the model Analog.Basic.Conductor which implements conductance as:

$$G_{actual} = \frac{G}{(1 + \alpha (T_{hp} - T_{ref}))} \quad (6)$$

While this formula can be used within a Calculus exam, the technique is applicable on formulas like (5).

### 6.1 Classical AD techniques

The fundamental terminologies and concepts of classical AD techniques for assignment-based procedural languages have been largely developed decades ago. Application of classical AD techniques is not best suitable to be applied on equation-based languages due to [15, 13]:

- Classical AD techniques are mainly designed for explicit assignments and not implicit equations
- Excessive number of expressions evaluations is performed
- Excessive storage for temporary variables and their gradient computations is needed

In order to overcome these drawbacks, an equation-based technique relying on fundamental notions of equation-based languages has been designed.

### 6.2 Equation-based AD technique

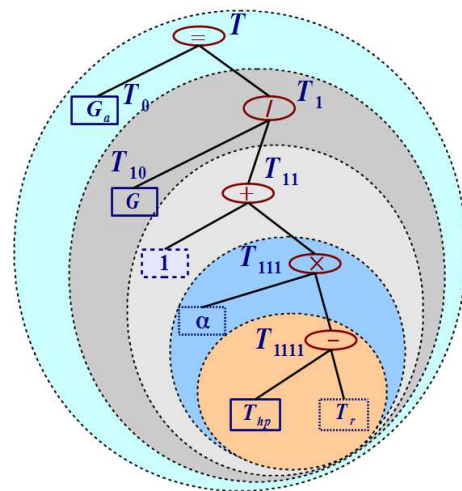


Figure 2: The AST of Equation 6 and the enumeration of expression subtrees

### Preprocessing: Intermediate elementary equations

The key idea of enabling differentiation of a complex equation is to decompose it into *elementary equations*. Each equation is composed of a unitary operator (e.g. +, -), a binary operator (e.g. +, -, \*,  $\Pi$ ) or an intrinsic function (e.g. sin, cos). By constructing the abstract syntax tree (AST) of the equation, elementary equations can be computed. In Figure 2, non-leaf nodes correspond to operands. Leaf nodes correspond to identities. Identities are distinguished according to their variability, e.g. constants, parameters and variables. Each subtree  $T_k$  is intuitively indexed with a binary number  $k$  according to its position in the tree. Each node with index  $k$  has children of index  $2k$  and  $2k + 1$ , if any. Intermediate equations and their derivatives can be easily computed via a left-right-node (LRN) traversal as follows:

$$\begin{aligned}
 T_{1111} &= T_{hp} - T_{ref} \\
 \implies T'_{1111} &= T'_{hp} - T'_{ref} \\
 T_{111} &= \alpha T_{1111} \\
 \implies T'_{111} &= \alpha' T_{1111} + \alpha T'_{1111} \\
 1/T_{11} &= 1 + T_{11} \\
 \implies T'_{11} &= -T'_{111} T_{11} T_{11} \\
 T_1 &= G T_{11} \\
 \implies T'_1 &= G' T_{11} + G T'_{11} \\
 \implies G'_a &= T'_1
 \end{aligned}$$

where

$$T'_k = \left( \frac{\partial T_k}{\partial p_1}, \frac{\partial T_k}{\partial p_2}, \dots, \frac{\partial T_k}{\partial p_m} \right)$$

Here, only intermediate equations corresponding to non-leaf nodes are considered. Moreover, Modelica capabilities are utilized for providing specialized treatment of the division operator, the most expensive arithmetic operator. This is done in a way that the let derivatives don't include further divisions.

### Processing: Accumulation

The previous intermediate equations though represent desired partial derivatives, however it requires a lot of storage for  $T'_k$ . To overcome this drawback one can rather iteratively accumulate the partial derivatives, within the same LRN traversal of the AST. On a manual basis, this corresponds to an iterative process of copy and paste of the intermediate equations one after another. In this way, the *accumulated intermediate*

*equations* become:

$$\begin{aligned}
 T_{1111}^{/*} &= T'_{hp} - T'_{ref} \\
 T_{111}^{/*} &= \alpha' T_{1111} + \alpha (T'_{hp} - T'_{ref}) \\
 T_{11}^{/*} &= -(\alpha' T_{1111} + \alpha (T'_{hp} - T'_{ref})) T_{11} T_{11} \\
 T_1^{/*} &= G' T_{11} \\
 &+ G (-(\alpha' T_{1111} + \alpha (T'_{hp} - T'_{ref})) T_{11} T_{11}) \\
 \implies G'_a &\equiv T_1^{/*}
 \end{aligned}$$

Now, it is enough to declare only one derivative object for  $G_a$ .

### Postprocessing: Common subexpressions

The accumulated intermediate equations contain multiplicative term that are going to be evaluated  $m$  times. Within the same LRN-traversal, these multiplicative terms (e.g.  $\alpha T_{11} T_{11}$ ) can be rather stored in an a local variable. These terms are stored in local variables `adl_*` within the `Conductor` model:

Listing 10: The AD version of the `Conductor` model

```

model Conductor
  extends ...;
  ...
protected
  Real T_1111;
  Real T_111;
  Real D_11;
  Real adl_11_1;
  Real adl_11_2;
  Real adl_11_3;
  Real adl_1_1;
  Real adl_1_2;
equation
  T_1111 = T_heatPort - T_ref;
  T_111 = alpha * T_1111;
  1/D_11 = 1 + T_111;
  adl_11_1 = - D_11 * D_11;
  adl_11_2 = adl_11_1 * alpha;
  adl_11_3 = T_111 * adl_11_1;
  adl_1_1 = G * adl_11_3;
  adl_1_2 = G * adl_11_2;
  g_G_actual[:] = g_G[:] * D_11 +
    adl_1_1 * g_alpha[:] + adl_1_2
    * (g_T_heatPort[:] - g_T_ref[:]);
  g_i[:] = g_G_actual[:] * v
    + G_actual * g_v[:];
  g_LossPower[:] = g_v[:] * i
    + v * g_i[:];
end Conductor;

```

This is an optional step, particularly, if the used Modelica compiler is capable of recognizing common subexpressions and treating them adequately. However this step still can be used for coming up with an

efficient compact representation of the partial derivatives.

## 7 Illustrative Example: The top model

Having performed equation-based AD of the Electrical.Analog sub-package, then parameter sensitivities of the electrical circuit in Analog.Basic.Example.ChuaCircuit are implicitly represented by simple slight modification:

- reimporting the AD version of the library and
- specifying the input Jacobian

Listing 11: The AD version of the ChuaCircuit model

```

model ChuaCircuit
  "Chua's circuit, ns, V, A"
  // import Modelica.Electrical.
    Analog.Basic;
  import ADMSL.Electrical.Analog.Basic;
  // import Modelica.Electrical.Analog.
  //   Examples.Utilities;
  import ADMSL.Electrical.Analog.
    Examples.Utilities;
  import Modelica.Icons;
  extends Icons.Example;
  import ADMSL.Utilities.*;
  inner parameter Integer NG = 8;
  Basic.Inductor L(L=18,
    g_L=unitVector(1,NG));
  Basic.Resistor Ro(R=12.5e-3,
    g_R=unitVector(2,NG));
  Basic.Conductor G(G=0.565,
    g_G=unitVector(3,NG));
  Basic.Capacitor C1(C=10, v(start=4),
    g_C=unitVector(4,NG));
  Basic.Capacitor C2(C=100,
    g_C=unitVector(5,NG));
  Utilities.NonlinearResistor Nr(
    Ga(min=-1) = -0.757576,
    g_Ga = unitVector(6,NG),
    Gb(min=-1) = -0.409091,
    g_Gb = unitVector(7,NG),
    Ve=1,
    g_Ve = unitVector(8,NG));
  Basic.Ground Gnd
equation
  // same as before
  ...
end ChuaCircuit;
    
```

Figures 3 and 4 show some results of the parameter sensitivities.

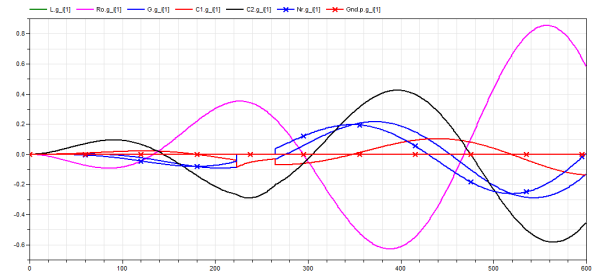


Figure 3: The sensitivities of the current at all components (i.e. L.i, Ro.i, G.i, C1.i, C2.i, Nr.i, Gr.i) w.r.t. the inductance L.L

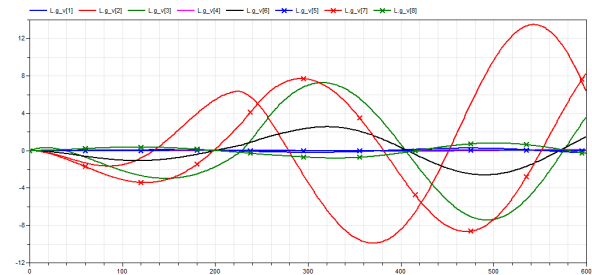


Figure 4: The sensitivities of L.v w.r.t. all parameters (i.e. L.L, Ro.R, G.G, C1.C, ... etc.)

## 8 Summary and outlook

In this paper, an equation-based methodology for AD of Modelica libraries has been comprehensively demonstrated. The new equation-based AD techniques have been designed for the Modelica language. With few simple code generation rules, parameter sensitivities of base models are additionally described. These rules make use of the art of AD to let modelers themselves be capable of manually differentiating large-set of models. Once a library is already differentiated for once, it can be used forever. The whole work serves as an experimentation platform for the implementation of AD tool. Already a lot of the functionalities are implemented within the tool ADModelica [14].

Further ongoing works are running on several dimensions:

- Ensuring that all used terminology is conform to the Modelica specification
- choosing the most proper constructs allowing further extension of this library to be compilable with arbitrary simulation environments

- Implementing unit tests not only for single components but also to ensure the correctness of intermediate components, processed and postprocessed classes
- Implementing further useful sensitivity-related functionalities, e.g. scaled sensitivities, finite difference functions, etc.
- Utilizing Modelica capabilities for operator overloading

Finally, there are further two issues that are worth to mention. First, although normal numerical solvers not especially designed for sensitivity analysis are used, the expensive computation of parameter sensitivities can be reduced by following a numerical integration approach based on Dicknson [8]. Smaller subsets of parameter sensitivities are considered one after another rather than considering the whole SES [9]. The structure of the proposed gradient-oriented code easily suggests that. Moreover, this approach is parallelizable in a scalable manner [22].

Second, the presented approach in this work is restricted to continuous-time based components by which large set of already existing libraries can be considered. However, theorems and concepts for handling discontinuous functions and hybrid systems already exist [18], out of which further extensions to this work can be considered.

## A Some modified components in the Modelica.Electrical.Analog library

```
ADMSL.Electrical.Analog.Interfaces.
Bases.OnePort
```

The TwoPin model with the MSL is slightly modified by letting the declared connectors become replaceable as follows:

Listing 12: Implementation of OnePort

```
partial model OnePort
  "Component with two electrical pins"
  Modelica.SIunits.Voltage v
  "Voltage drop between the two pins";
  Modelica.SIunits.Current i
  "Current flowing from pin p to pin n";
  replaceable Modelica.Electrical.Analog.
    Interfaces.PositivePin p
    constrainedby
      Modelica.Electrical.Analog.
```

```
    Interfaces.PositivePin;
  replaceable Modelica.Electrical.Analog.
    Interfaces.NegativePin n
    constrainedby
      Modelica.Electrical.Analog.
        Interfaces.NegativePin;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

```
ADMSL.Electrical.Analog.Interfaces.
Bases.TwoPin
```

The TwoPin model with the MSL is slightly modified by letting the declared connectors become replaceable as follows:

Listing 13: Implementation of two pins

```
partial model TwoPin
  "Component with two electrical pins"
  Modelica.SIunits.Voltage v
  "Voltage drop between the two pins";
  replaceable Modelica.Electrical.Analog.
    Interfaces.PositivePin p
    constrainedby
      Modelica.Electrical.Analog.
        Interfaces.PositivePin;
  replaceable Modelica.Electrical.Analog.
    Interfaces.NegativePin n
    constrainedby
      Modelica.Electrical.Analog.
        Interfaces.NegativePin;
equation
  v = p.v - n.v;
end TwoPin;
```

## References

- [1] The ADMSL library. <https://github.com/AIT-CES-LAB/ADMSL>.
- [2] Wolfram SystemModeler. <http://www.wolfram.com/system-modeler/>.
- [3] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem. *Computers and Chemical Engineering*, 34(11):1737–1749, 2010.
- [4] J. Andersson, B. Houska, and M. Diehl. Towards a computer algebra system with automatic differentiation for use with object-oriented modelling languages. In *EOOLT'2010: The 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Oslo, Norway, Oct. 2010.

- [5] I. Bauer, H. G. Bock, S. Körkel, and J. P. Schlöder. Numerical methods for optimum experimental design in DAE systems. *Journal of Computational and Applied Mathematics*, 120:1 – 25, 2000.
- [6] C. H. Bischof, H. M. Bücker, W. Marquardt, M. Petera, and J. Wyes. Transforming equation-based models in process engineering. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Implementations*, Lect. Notes in Comp. Sc. and Eng., pages 189–198. Springer, 2005.
- [7] W. Braun, L. Ochel, and B. Bachmann. Symbolically derived Jacobians using automatic differentiation - enhancement of the OpenModelica compiler. In *Modelica'2011: The 8th International Modelica Conference*, Dresden, Germany, Mar. 2011.
- [8] R. Dickinson and R. Gelinas. Sensitivity analysis of ordinary differential equation systems - a direct method. *Journal of Computational Physics*, 21:123–143, 1976.
- [9] A. Elsheikh. *Modelica-based computational tools for sensitivity analysis via automatic differentiation*. Dissertation, RWTH Aachen university, Aachen, Germany, 2011.
- [10] A. Elsheikh. ADGenKinetics: An algorithmically differentiated library for biochemical networks modeling via simplified kinetics formats. In *Modelica'2012: The 9th International Modelica Conference*, number 076 in Linköping Electronic Conference Proceedings, pages 915 – 926, Munich, Germany, Sep. 2012.
- [11] A. Elsheikh. Assisting identifiability analysis of large-scale dynamical models with decision trees: DecTrees and InteractiveMenus. In *EuroSim'2013: The 8th EUROSIM Congress on Modelling and Simulation*, pages 300 – 305, Cardiff, Wales, UK, Sep. 2013.
- [12] A. Elsheikh. Derivative-based hybrid heuristics for continuous-time simulation-optimization. *Simulation Modelling Practice and Theory*, 2013. In Press.
- [13] A. Elsheikh. An equation-based algorithmic differentiation technique for differential algebraic equations. *Computational and applied Mathematics*, 201x. Submitted. Available online as a technical report.
- [14] A. Elsheikh, S. Noack, and W. Wiechert. Sensitivity analysis of Modelica applications via automatic differentiation. In *Modelica'2008: The 6th International Modelica Conference*, volume 2, pages 669–675, Bielefeld, Germany, March 2008.
- [15] A. Elsheikh and W. Wiechert. Automatic sensitivity analysis of DAE-systems generated from equation-based modeling languages. In C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 235–246. Springer, 2008.
- [16] A. Elsheikh and W. Wiechert. Accuracy of parameter sensitivities of DAE systems using finite difference methods. *Mathematical Modelling, IFAC Papers Online*, 7(1):136 – 142, Feb. 2012. Presented in MATHMOD'2012, The 7th Vienna International Conference on Mathematical Modelling, Vienna, Austria.
- [17] A. Elsheikh and W. Wiechert. A structure-preserving approach for sensitivity analysis of higher index differential algebraic equations. *Mathematics and Computers in Simulation*, 201x. Submitted.
- [18] S. Galán, W. F. Feehery, and P. I. Barton. Parametric sensitivity functions for hybrid discrete/continuous systems. *Applied Numerical Mathematics*, 31(1):17 – 47, 1999.
- [19] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.
- [20] A. K. Gupta and S. A. Forth. An AD-enabled optimization toolbox in LabVIEW™. In S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors, *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*, pages 285–295. Springer-Verlag Berlin Heidelberg, 2012.
- [21] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, Sept. 2005.
- [22] X. Ke. Tools for sensitivity analysis of modelica models. Master's thesis, Siegen University, Germany, 2009.
- [23] S. Li, L. Petzold, and W. Zhu. Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem. *Applied Numerical Mathematics: Transactions of IMACS*, 32(2):161–174, Feb. 2000.
- [24] U. Naumann. *The art of Differentiating Computer Programs, an Introduction to Algorithmic Differentiation*. SIAM, 2012.
- [25] H. Olsson, H. Tummescheit, and H. Elmqvist. Using automatic differentiation for partial derivatives of functions in Modelica. In *Modelica'2005: The 4th International Modelica Conference*, Hamburg, Germany, 2005.
- [26] M. Sjölund and P. Fritzson. An openmodelica java external function interface supporting metaprogramming. In *Modelica'2009: The 7th International Modelica Conference*, Como, Italy, 2009.
- [27] W. Wiechert and R. Takors. *Validation of Metabolic Models: Concepts, Tools, and Problems*. Taylor & Francis, 2004.

# Modelica Based Parser Generator with Good Error Handling

Arunkumar Palanisamy<sup>1</sup>, Adrian Pop<sup>1</sup>, Martin Sjölund<sup>1</sup>, Peter Fritzson<sup>1</sup>

<sup>1</sup>PELAB – Programming Environment Laboratory  
Department of Computer and Information Science  
Linköping University, SE-581 83 Linköping, Sweden  
{arunkumar.palanisamy, adrian.pop, martin.sjolund, peter.fritzson}@liu.se

## Abstract

This paper describes the new OpenModelica Compiler-Compiler (OMCC) including a parser generator, OMCCp which is based on an LALR parser generator extended with advanced error handling facilities. It is implemented in the MetaModelica language with parsing tables generated by the tools Flex and Bison. It is integrated with the MetaModelica semantics specification language, based on operational semantics for generating executable compiler and interpreter modules.

The OMCCp parser generating part of OMCC is being used for the full Modelica language grammar as well as for the language extensions of MetaModelica, ParModelica, and Optimization specifications. The generated parsers have reasonable performance compared to other parser generators.

*Keywords:* Modelica, MetaModelica, Flex, Bison, ParModelica, Optimization, OMCCp

## 1 Introduction

The OpenModelica environment currently makes use of the tool ANTLR (Another tool for Language Recognition) [11] to generate the parser for the OpenModelica Compiler (OMC). In this paper we present an alternative to ANTLR, the new OpenModelica Compiler-Compiler parser generator (OMCCp), developed within the OpenModelica project. The tool is implemented in MetaModelica which is an extension of the Modelica language for modeling the semantics of languages. The work [9] [11] is integrated with the recently developed bootstrapped OpenModelica compiler (OMC) [14].

The ANTLR parser generator which has been used in the OpenModelica project for several years has well

known disadvantages including memory overhead, bad error handling, and lack of type checking. Also it does not generate directly MetaModelica code for building the Abstract Syntax Tree (AST).

Since the AST nodes are initially generated by C functions (for later conversion into MetaModelica garbage collected memory space) without strong type checking in the C language, small errors in the semantic actions of the grammar are not detected at generation time and can give rise to hard-to-find errors in the generated code (even small errors in the grammar actions C code lead to segmentation faults).

When the semantic actions can be specified in MetaModelica and the AST builder directly generates the MetaModelica code, the above mentioned errors can be completely eliminated.

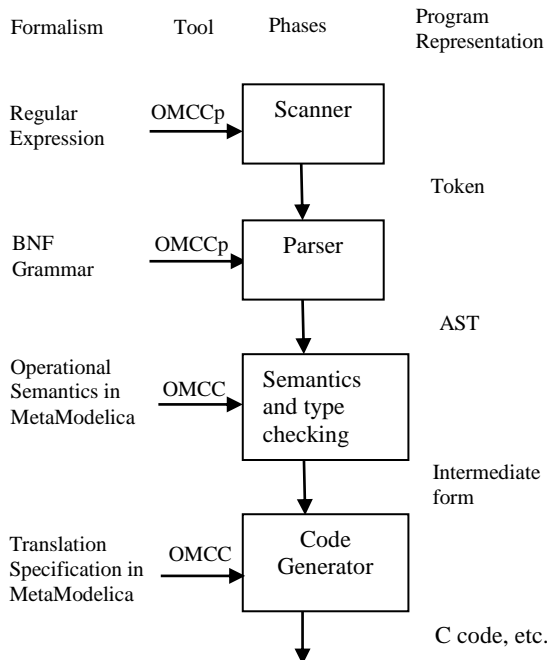
The need for good error handling as well as avoidance of certain AST-building errors has motivated the development of the Modelica based parser generator with good error handling [9][11].

This paper is structured as follows: Section 2 describes the different steps involved in designing a compiler from different specification formalism. Section 3 presents the error-handler features added to the tool and also illustrates the different types of error handler messages displayed to the user in the case of erroneous programs. Section 4 explains the main MetaModelica language constructs used in this implementation for generating the Abstract Syntax Tree (AST). Section 5 explains the OMCCp tool design and architecture and Section 6 presents test results and performance measurements. Finally Section 7 concludes the paper summarizing achieved results.

## 2 Background

### 2.1 Generating Compiler Phases

A compiler can be generated from a formal specification in different formalisms, as depicted in Figure 1.



**Figure 1.** Design stages of a compiler from different specification formalism

Figure 1 describes the stages involved in generating a compiler from specifications in different formalisms. Generally a compiler is divided into two parts: the front-end and the back-end. The Scanner and Parser constitute the front-end phase whereas Optimization and Code generator constitute the back-end phase of the compiler. In this paper we focus on the front-end parts of the compiler.

### 2.2 Lexical Analysis

The Lexical analysis, performed by a scanner, is the first stage of the compilation process. It receives the source code as input and generates tokens. It identifies the special tokens defined by a language making it simpler for the next phase of the compiler. The tokens are usually specified by using Regular Expressions.

There are several tools available that automate the process of constructing the transition rules to identify the tokens for a scanner. We use the Flex tool [13] for this purpose which generates C code; the generated C code is later used by OMCCp to generate the appropriate lexer components in MetaModelica [1] [9] [11].

### 2.3 Syntax Analysis

The syntax analysis (also called parsing) is the second stage of the compilation process. The parser takes the tokens generated by the lexer and determines whether the tokens are constructed according to the rules of the grammar. During this process the Abstract Syntax Tree (AST) is created if the input conforms to the defined grammar. Otherwise an error message is reported.

The AST is used as input to the back-end. The back-end uses the AST for type checking, optimization and finally generates machine specific code. The grammar rules are usually specified in the form of BNF (Backus-Naur Form). We use the popular Bison tool for writing the grammar rules; the generated tables are used by OMCCp as part of the parser algorithm written in MetaModelica that interprets these parse tables [1][9][11].

## 3 Error Handling

The error handling techniques in the front-end are more relevant during the Syntax analysis phase than in the lexical analysis phase. Only a few errors can be detected by the lexical analysis, such as non-terminated comments, use of invalid characters, or unrecognized tokens. One possible error-recovery strategy implemented in a lexer is to ignore invalid characters from the input and continue the process.

The error handling techniques can be divided into two categories: Error recovery techniques and error message display. Error recovery techniques are concerned with how the parser can keep parsing after an erroneous token is found. Error message display concentrates on how to present useful hints for the developer in order to correct the source code. In this section we will present the two topics for error handling during the syntax analysis phase [1] [9] [11] [2] [4].

### 3.1 Error Recovery

Error recovery techniques try to improve the quality of the parser by different techniques such as primary recovery or secondary recovery. The first condition to start the recovery is to access the configuration obtained when the token preceding the error token was shifted onto the stack

Primary recovery techniques are related to single token modification from the list of tokens. Single modification is only possible when the error is classified as simple. Such a modification can be insertion, deletion, substitution, or merging.



Each attempt to perform a repair is known as a trial. A common technique for searching the trials is to attempt to repair the error token by performing one of these operations: *merging*, *insertion*, *substitution*, *scope recovery* and finally *deletion*. In the case of insertion or substitution a set of possible candidates should be generated and then from there a single candidate or none should be selected.

When the error requires more than a simple modification, the list of tokens needs to be reduced. This can be done by discarding tokens that precedes or follows the error token. This is known as secondary recovery [1] [9] [11] [2].

## 3.2 Error Messages

OMCCp uses a primary recovery technique to display all the possible recovery candidates to the developer. When an error is found, the parser fires the error handler function including the environmental variables that contain the actual configuration of the parser and the backed up configuration when the last token was shifted. This backup is used to start an extensive search for the possible valid tokens to be replaced. This allows the developer to better understand the messages and makes it easier to select the correct token.

OMCCp uses seven different kinds of error messages for the syntax analysis and one more for the lexical analysis. The error messages displayed in this implementation are discussed below.

### 3.2.1 Erase Token

The *erase token* message occurs when the parser finds same token repeated more than once. To test if a token can be erased, the parser is run on the remaining list of tokens ignoring the current token. If the test succeeds a proper error message is displayed to the user suggesting a possible solution for erasing the repeated token [9] [11]. An example error message display is given below.

```
[../../../../testsuite/omcc_test/error5.mo:10:20-10:24:writable]
Error: Syntax error near: 'if x == 10 then then', REPLACE token with '+' or '-' or '.' or 'NOT', ERASE token 'then'
```

### 3.2.2 Insert Token

To test if a token can be inserted before the error token, the parser is run on a modified list of the remaining tokens by placing the candidate token before the error token as current token. The candidate token is selected if the test succeeds and placed in the candidate list [9] [11]. If there are items in the candidate list we display proper message to the user. An example error message is given below.

```
[../../../../testsuite/omcc_test/error3.mo:9:3-9:4:writable]
Error: Syntax error near: 'if x == 10', INSERT token 'THEN'
```

### 3.2.3 Replace Token

The replace token is similar to insert token error message. The parser is run modifying the remaining list of tokens by placing the candidate token before the error token as current token. The candidate token is selected if the test succeeds and placed in the candidate list and a proper error message is displayed to the user [9] [11].

```
[../../../../testsuite/omcc_test/error2.mo:7:1-7:6:writable]
Error: Syntax error near: 'while x <> 99 then', 'THEN', REPLACE token with 'Loop'
```

### 3.2.4 Insert Token at End

This message is used only at the end of the program, when no other token is available in the input of tokens and a non-finished acceptance state has been achieved. All the tokens are tested to verify if they can make the program to end in a valid acceptance state. If a token succeeds then the proper message is displayed to the user as a possible solution to fix the error [9] [11]. An example message is given below.

```
[../../../../testsuite/omcc_test/error6.mo:14:1-14:15:writable]
Error: Syntax error near: 'end error_test', INSERT at the End token 'SEMICOLON'
```

### 3.2.5 Merge Token

Sometimes a space can be inserted by mistake between two tokens and make a keyword appear as two separate identifier tokens. In this case the error token and the token that follows it are processed again by the Lexer with their value concatenated.

If the lexer combines them as a valid token this token is tested to see if it satisfies the test and is a valid configuration for the parser [9] [11]. If it succeeds the system displays a proper message to the user. An example error message is given below.

```
[../../../../testsuite/omcc_test/error4.mo:10:9-10:10:writable]
Error: Syntax error near: 'if x = = 10 then', MERGE tokens '=' and '='
```

### 3.2.6 Generic Error

It is possible that no solution or candidate is found for the current error token. In these cases a generic error message is displayed to the user without any further description of the error than the location of the token.

In this situation the developer needs to fix the error token without any hint [9] [11].

### 3.2.7 Custom Error Message

While designing a grammar it is sometimes necessary to communicate to the developer that a certain transformation rule must not be used. A more clear language than presented by the error messages above should be used. For this reason the possibility of a custom error message has been introduced. Such an error message is added in the grammar rules. The use of this custom message will activate the error flag in the parser and will start the simple error recovery technique.

## 4 MetaModelica

MetaModelica is an extension of the Modelica language which can be used to model the semantics of languages, specify symbolic transformations, etc. It is based on the operational semantics language specification formalism; a rule in operational semantics becomes a case within match-expressions [6] [7].

### 4.1 Uniontype

A `uniontype` in MetaModelica is a collection of one or more record types. It can be recursive and can include other uniontypes. It is mainly used for the construction of Abstract Syntax Trees (AST) [6] [7].

Example:

```
uniontype Exp
  record INT
    Integer integer;
  end INT;
  record BINARY
    Exp exp1;
    BinOp binOp;
    Exp exp2;
  end BINARY;
end Exp;
```

### 4.2 Match Expressions

The `match` expression is similar to switch statements in C but even closer to match in functional programming languages with some extra features.

The `match` expression can return more than one value and supports pattern matching. The `match` construct contains case blocks. Each `case` can contain an equation block. The program flow tries to execute one instruction after the other in a specific local equation block. If an instruction is not executed or failed the next case is tried until a match is found. The wildcard ‘`_`’ (underscore) can be used to match all the cases [6] [7].

For example

```
function eval
  input Exp inExp;
  output Integer outInteger;
algorithm
  outInteger := match(inExp)
  local
    Integer ival, v1, v2, v3;
    Exp e1, e2, e;
  case INT(ival) then ival;
  case BINARY(e1, binop, e2)
  equation
    v1 = eval(e1);
    v2 = eval(e2);
    v3 = applyBinop(binop, v1, v2);
  then
    v3;
  case UNARY(unop, e)
  then v2=applyUnop(unop, eval(e));
  end match;
end eval;
```

In the above example we can see that the function `eval` contains one input formal parameter of type `Exp` and one output formal parameter of type `Integer` followed by the `match`-expression which tries to match any of the three cases according to the user input.

The first case results in the evaluation of `INT` record constructor node applied to an integer. The second case results in evaluation of a binary operator node `binary` to `v3`, if `v3` is the result of successfully applying the binary operator to `v1` and `v2`, which is the evaluated result of its children `e1` and `e2`. The third case results in the evaluation of unary operator node `UNARY` to `v2`.

### 4.3 List

The `List` constructor is used to create linked list structures. Lists are used in modeling of flexible variable-length collections of symbolic elements. The operand `::` is used to add an element at the front of a list (or, depending on the context, retrieve by matching) an element from the list [6] [7].

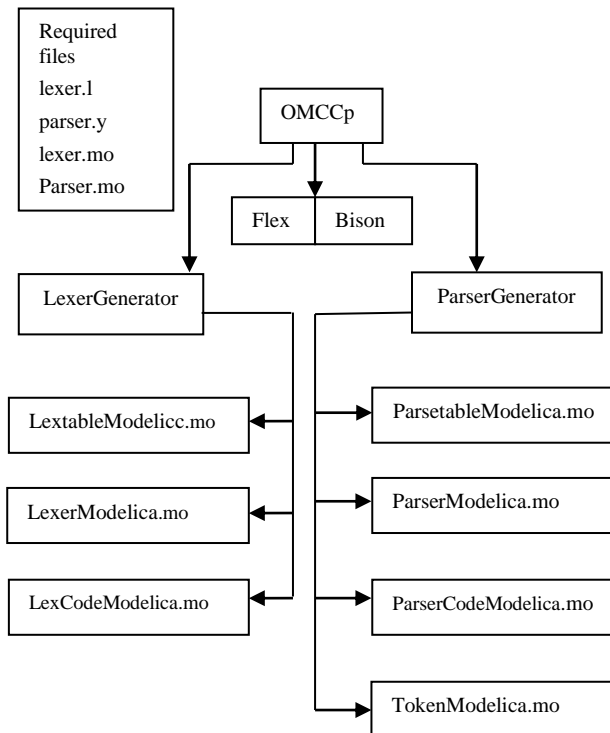
Example

```
list<Integer> a= {1,2,3};
i::a=a;
a=i::a;
```

The first line creates a list `a` of integers, the second line is used for the retrieve operation; it takes the top element 1 from the list and stores it in the variable `i` and stores the remaining list in variable `a`. The third line performs the add operation which adds an item `i` into the list `a`.

## 5 OMCCp DESIGN

The design architecture of OMCCp with the lexer and parser components is depicted in Figure 2.



**Figure 2.** OMCCp (OpenModelica Compiler-Compiler) Lexer and Parser Generator

The lexer and parser are generated based on C files generated by the tools Flex and Bison from the grammar files `lexer.l` and `parser.y`.

The generated C file contains three main parts which constitutes the lexer and parser. We can identify: a section with the transition arrays, a section with code that runs the algorithm for the lexer or parser and finally an action resolution section which contains the returns tokens actions for the lexer and reduce operation which constructs the AST.

Based on the identification of the main parts, the OMCCp design is divided into two parts namely the `LexerGenerator` and the `ParserGenerator` which are responsible for generating the necessary Lexer and Parser components in `MetaModelica`.

### 5.1 LexerGenerator

The `LexerGenerator` is the main package of the lexer generator which generates the three necessary lexer packages in `MetaModelica`, namely `LexerModelica.mo`, `LextableModelica.mo` and `LexcodeModelica.mo`.

### 5.1.1 Lexer.mo

`Lexer.mo` is the main file which contains the calls to other functions in `Lextable.mo` and `LexCode.mo` that constitutes the lexer. The main function of this package is to load the source code file and recognize all the tokens described by the grammar.

To recognize the token the lexer.mo runs DFA (Deterministic Finite Automata) based on the transition arrays found in `Lextable.mo`. When it reaches an acceptance state it calls the function `action` in `LexCode.mo` which returns list of tokens that are input to the parser. The interface of the function which performs this operation is given below.

```

function scan
  input String fileName "input source
  code file";
  input list<Integer> program "source
  code as a stream of Integers";
  input Boolean debug "flag to activate
  the debug mode";
  output list<OMCCTypes.Token> tokens
  "return list of tokens";
end scan
  
```

### 5.1.2 LexTable.mo

`LexTable.mo` is the source file for `Lexer.mo` which contains the transitions arrays for performing transitions to new states and finding the tokens from the input stream.

### 5.1.3 LexCode.mo

`LexCode.mo` contains all the specific actions that a lexer performs when a token is recognized. There are three types of action a lexer can perform: *ignore* token, *return* specific token, and *switch* to another DFA.

The first action, ignore token, is performed by the lexer when a space, line feed, or block of comment has been found in the input stream. The tokens ignored by the lexer simplify the task of the parser as these tokens will not be used in the construction of the grammar.

The second action returns a specific token when a token is recognized by the lexer, the code of the action determines the tokens to be returned.

The third action is to switch from one DFA to another. This operation is performed in a few situations for example when the DFA finds a starting comment block `/*`, and all the subsequent tokens are required to be ignored or categorized as a different token, e.g. in case of determining a string. For this action a new startup state is set in the machine and new characters are processed by a different DFA than the original.

After the end of tokens is reached, i.e., ‘\*/’ then the start state returns to the original one.

## 5.2 Parser Generator

The parser generator is the main package of the parser that performs the syntax analysis of the compiler. This package generates four packages comprising the parser in MetaModelica namely TokenModelica.mo, ParserTable.mo, ParserCodeModelica.mo and ParserModelica.mo.

### 5.2.1 Parser.mo

The main function of this package is to efficiently convert the list of tokens received by the Lexer into Abstract Syntax Tree (AST). The package also contains the implementation of the LALR algorithm. For performing this task the package uses the parse table located in the package ParseTable.mo, to perform the shift-reduce action calls it uses the package ParserCodeModelica.mo. The interface of the function which starts the construction of the AST is given below.

```
function parse "realize the syntax
  analysis over the list of tokens and
  generates the AST tree"
input list<OMCCTypes.Token> tokens "list
of tokens from the lexer";
input String fileName "file name of the
source code";
input Boolean debug "flag to output
debug messages that explain the states
of the machine while parsing";
output Boolean result "result of the
parsing";
output ParseCode.AstTree ast "AST tree
that is returned when the result output
is true";
end parse;
```

### 5.2.2 ParseTable.mo

ParseTable.mo contains the arrays that allow the Parser package to run the Push down Automata (PDA) and perform the shift-reduce action which constructs the AST.

### 5.2.3 ParseCode.mo

The package ParseCode.mo contains the specific Reduce operations that each grammar performs when a certain rule matches the input tokens. The main function of this file is to handle the MultiTypedStack that is used by the parser to construct the AST. The MultiTypedStack handles the reduce operations requested by the LALR parsing algorithm. The MultiTyped stack contains one stack for each type found in the grammar specification and is defined in the MetaModelica language. An example interface is presented.

```
uniontype AstStack
record ASTSTACK
list<Absyn.Exp> stackExp;
list<String> stackString;
list<Integer> stackInteger;
end ASTSTACK;
end AstStack;
```

When the parser finds a Shift operation it calls the ‘function push’ on this file which will push a String value into the stackString. During the reduce operation the parser needs to know which stack to use for each of the constructions of the AST. The way this MultiTypedStack works can be explained in the following example that presents the reduce operation, the build of the AST operation and finally a push back into the stack which builds the AST.

```
case (82,_) // #line 413
"parserModelica.y"
equation
// reduce
(info, skToken) =
getInfo(skToken,mm_r2[act]);
v2Comment::skComment = skComment;
v1Algorithm::skAlgorithm =
skAlgorithm;
// build
vAlgorithmItem
=Absyn.ALGORITHMITEM((v1Algorithm),SOME((v
2Comment)),info);
// push Result
skAlgorithmItem=
vAlgorithmItem::skAlgorithmItem;
then ();
```

In the above case the reduction takes three items from the three different stacks and constructs an Absyn.ALGORITHMITEM object. After this it pushes the result back into the stack for the Absyn.Algorithm type called skAlgorithmItem. Another feature that can be useful for the reductions is the use of the info keyword. In the example, we can see that the instruction uses a stack called skToken to retrieve the token information.

The token information returns an info token of type Absyn.Info which contains the combined information of the first and the last token in the stack that are used for this reduction. This makes it possible for the developer to insert information about the location that can be used later in the other phases of the compiler. The package also contains another important function getAst which returns the result of AST tree to the parser. The interface of the function is given below.

```
function getAST "returns the AST built by
the parser"
input AstStackastStk aststack
"MultiTypedStack used by the parser";
output AstTree ast "returns the AST in
the final type of the tree";
end getAST;
```

### 5.2.4 Token.mo

The package contains the complete list of tokens used by the grammar with their respective codes. This file is the link between lexer and parser; it is used by both the lexer and parser to identify the tokens in the same way.

When the parser receives the token code from the lexer it performs a translation into local codes that are only used by the parser to simplify the addressing in predictive arrays. Each token is defined as an Integer-type constant. The interface of this package is presented below.

```
package TokenModelica
  constant Integer T_ALGORITHM = 258;
  constant Integer T_AND = 259;
  constant Integer BLOCK = 261;
  constant Integer CLASS = 262;
  constant Integer CONNECT = 263;
end TokenModelica
```

## 6 Testing

We performed testing using Modelica files from the OpenModelica testsuite. We also ensured that the tests covered full Modelica and MetaModelica grammar proving the correctness of the parser.

In the later stages of testing we also covered the ParModelica and Optimization grammars. During the testing we did not face any memory overhead problems and also measured the performance of the parser including AST building. The performance times are reasonable when compared to ANTLR. An example of sample test case of correct and an erroneous model with the respective output message are presented.

#### A Sample of correct input

```
model Circle
  Real x_out;
  Real y_out;
  Real x(start=0.1);
  Real y(start=0.1);
equation
  der(x) = -y;
  der(y) = x;
  x_out = x;
  y_out = y;
end Circle;
```

This input will generate the following success message

```
Parsing Modelica with file
../omcc_test/Test1.mo
// SUCCEED
// args:../omcc_test/Test1.mo
```

This sample of erroneous input will on the other hand generate a parse error.

```
class error_test
  int x,y,z,w;
  algorithm
  while x <> 99
    x := (x+111) - (y/3);
    if x == 10 then
      y := 234;
    end if;
  end while;
end error_test;
```

This input will cause the following error message to be emitted by an OMCCp generated parser

```
Parsing Modelica with file
../omcc_test/error3.mo
[../testsuite/omcc_test/error3.mo:9:3-9:4:writable] Error: Syntax error near:
'while x <> 99', INSERT token 'LOOP'
args:../testsuite/omcc_test/error3.mo
```

The same input will cause the following error message to be emitted by an ANTLR generated parser

```
Loaded all files without error
>true
"
""
{fail() }
```

Take a look at a second erroneous sample input, an erroneous piece of MetaModelica code.

```
function add
  input Integer ininteger;
  input Integer ininteger1;
  output Integer outinteger;
  algorithm
    outinteger:=
      match(ininteger,ininteger1)
        local
          Integer a,b,c;
          case(a,b)
            equation
              c=a+b;
            then
              c;
          end matchcontinue;
        end add;
```

This will cause the following error message to be emitted using an OMCCp generated parser

```
Parsing Modelica with file
../omcc_test/error2.mo
[../testsuite/omcc_test/error2.mo:14:13-14:30:writable] Error: Syntax error near:
'end matchcontinue', REPLACE token with
'ENDMATCH'
args:../testsuite/omcc_test/error3.mo
```

It will cause the following error message to be emitted using an ANTLR generated parser.

```
Loaded all files without error
"true
"
""
{fail()}
```

Regard a third erroneous sample input

```
class error_test
  int x,y,z,w;
  algorithm
  while x <> 99 loop
    x := (x+111) - (y/3);
    if x == 10 then
      y := 234;
    end if;
  end while;
end error_test;
```

This will cause the following error message from an OMCCp generated parser.

```
Parsing Modelica with file
../omcc_test/error6.mo
[../../../../testsuite/omcc_test/error4.mo:6:9-6:10:writable]
Error: Syntax error near:'if x = = 10 then', MERGE tokens '=' and '='
```

and this error message using an ANTLR generated parser.

```
Loaded all files without error
"true
"
""
{fail()}
```

### 6.1 Time Performance

The performance measurements for the test cases have been done using the OpenModelica test server. The models are taken from the Modelica Standard Library (MSL 3.2.1) as well as from the OpenModelica test suite. A selection of measurements is listed in the following Table 1.

**Table 1.** Time measurement of OMCCp and ANTLR generated parsers on a set of test models.

Model	Size	No of Tokens	OMCCp (time ms)	ANTLR (time ms)
Dcmotor	2kB	172	3.1	0.45
Influenza	4kB	642	10.4	1.10
Test1	26kB	8443	84.3	6.90
Icon	42kB	9234	179.7	8.68
SIunit	94kB	14330	303.8	18.57
Electrical/Digital	366kB	71117	1576	93
Electrical/Machine	763kB	143505	3202	158
Test2	1MB	214617	4681.8	254
Total	11MB	2078841	51362	2704

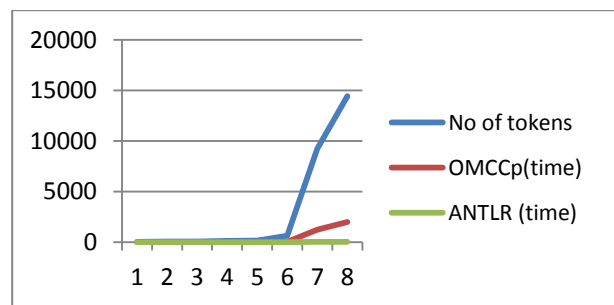
We can see that the ANTLR generated parser is about 5 to 16 times faster than OMCCp including AST building. The OpenModelica implementation of an ANTLR generated parser is extremely fast. The OMCCp parser performance is comparable to typical non-optimized parser generators.

The OMCCp generated parser is roughly 16 times slower than ANTLR for the very large Modelica models like `Total.mo` which contains the whole Modelica standard library 3.2.1 of 11 MB, but in general for smaller test cases the parser is only 6 times slower.

From the table we can also see that the OMCCp parser does not scale as well in the case of large models compared to ANTLR. Even though the ANLTR parser is faster, the OMCCp tool has several advantages compared to ANTLR. One should also note that the current implementation of OMCCp was not optimized and its performance can be further improved.

During testing we found that the OMCCp tool did not have any memory overhead problems when parsing a large number of test cases in a single attempt whereas the ANTLR parser had higher memory overhead when many test cases were parsed in a single attempt. Moreover, the OMCCp tool has better error handling features compared to the ANTLR tool, which makes it easier to use.

#### 6.1.1 Performance Graph



**Figure 3:** Graph representation of time performance of OMCCp with ANTLR

From the above graph we can see that the OMCCp parser keeps its timing performance close to that of ANTLR for a certain number of tokens recognized by the lexer, but when the number of tokens increases the performance gap increases between the tools.

Still the OMCCp tool has good performance when parsing grammars of size less than 100kB, with times of under a second. One of the reasons that the OMCCp is slower is that garbage collection is used in Meta-Modelica whereas in ANTLR the memory allocation/de-allocation is manually programmed.

## 7 Conclusion

In this paper we have presented a fully implemented OMCCp lexer and parser generator integrated with MetaModelica as a semantic specification language in the new bootstrapped OpenModelica compiler.

We have tested this tool on number of small languages as well as on the large Modelica grammar. The generated parsers offer good error handling and comprehensive error messages to the user. OMCCp can also be used as a parser generator for any language for which an LALR(1) grammar is available. The associated language semantics can be specified using MetaModelica.

The generated parsers are still a bit slow for production usage on large programs but we expect to improve the performance by further tuning.

## 8 Acknowledgments

This work has been partially supported by the Swedish Governmental Agency for Innovation Systems (Vinnova) within the ITEA2 MODRIO project, and by the Swedish Research Council (VR).

## References

- [1] Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman. *Compilers Principles, Techniques, and Tools*, Second Edition. Addison-Wesley, 2006.
- [2] Rober Bilos. *Syntactic Error Diagnosis and Recovery*. Master Thesis, Linköping University, Department of Computer and Information Science. 1983.
- [3] Michael Burke and G.A. Fisher Jr. A Practical Method for Syntactic Diagnosis and Recovery. In *Proceedings of the 1982 SIGPLAN symposium on Compiler constructions*, 1982.
- [4] Michael G. Burke and Gerald A. Fisher. A practical method for LR and LL Syntactic Error Diagnosis and Recovery. *ACM Transactions on Programming Languages and Systems*, March 1987.
- [5] Peter Fritzson. *Principles of Object-oriented modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.
- [6] Peter Fritzson, Adrian Pop and Martin Sjölund. *Towards Modelica 4 Meta-Programming and Language Modeling with MetaModelica 2.0*, Technical reports Computer and Information Science Linköping University Electronic Press, ISSN:1654-7233; 2011:10.
- [7] Peter Fritzson and Adrian Pop. *Meta-Programming and Language Modeling with MetaModelica 1.0*. Technical reports Computer and Information Science Linköping University Electronic Press, ISSN: 1654-7233. 2011:9.
- [8] Peter Fritzson et al. Compiler Construction laboratory assignments. Compendium, Bokakademin, Linköping University, Department of Computer and Information Science, 2011.
- [9] Edgar Alonso Lopez-Rojas. *OMCCp: A Meta-Modelica Based Parser Generator Applied to Modelica*. Master Thesis, Linköping University, Department of Computer and Information Science, PELAB- Programming Environment Laboratory, ISRN:LIU-IDA/LITH-EX-A--11/019--SE, May 2011.
- [10] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.6*, November 2010. <http://www.openmodelica.org>.
- [11] Arunkumar Palanisamy. Extended MetaModelica based Integrated Compiler generator. Master's-Thesis, Linköping University, Department of Computer and Information Science, PELAB-Programming Environment Laboratory, ISRN:LIU-IDA/LITH-EX-A--12/058--SE, October 2012.
- [12] Terence Parr and R W Quong. *ANTLR: A Predicated-LL(k) Parser Generator*. Software Practice Experience, 25(7):789, 1995. ISSN 00380644. URL <http://portal.acm.org/citation.cfm?id=213593.213603>.
- [13] Vern Paxson. Flex Manual, 2002. URL <http://flex.sourceforge.net/manual/>. [Accessed May 2011].
- [14] Martin Sjölund, Peter Fritzson, and Adrian Pop. Bootstrapping a Modelica Compiler aiming at Modelica 4. In *Proceedings of the 8th International Modelica Conference (Modelica'2011)*, Dresden, Germany, September 2011.





# Nonlinear inverse models for the control of satellites with flexible structures

Matthias J. Reiner    Johann Bals

German Aerospace Center (DLR), Institute of System Dynamics and Control  
82234 Wessling, Germany, {Matthias.Reiner, Johann.Bals}@DLR.de

## Abstract

Nonlinear inverse dynamic models can be utilized in various parts of advanced model-based control system design: reference trajectory optimization, feed-forward control and feedback linearization [35]. In this paper, a new synthesis approach for nonlinear inverse dynamic models of satellites with flexible structures is presented. For satellite configurations with unstable zero dynamics, a stable inverse model approximation is proposed which has been successfully applied to robots with flexible bodies.

This inverse modeling approach is part of the newly developed DLR Space Systems Library for object-oriented modeling and simulation of satellites and launchers in a detailed space environment. For satellites with flexible structures, the library provides models for normal simulation mode and the necessary tools to directly generate approximate inverse models.

In this paper, trajectory optimization is shown to be an important use case for inverse dynamic models. By inversion based reformulation of the trajectory optimization problem, the optimal reference motion of the control system can be determined in a reliable and efficient way.

*Keywords: satellite modeling; nonlinear inverse model; trajectory optimization; flexible structure*

## 1 Introduction

The Space Systems Library (SSL) was built to develop advanced control systems for satellites / spacecraft with flexible structures. In particular, one goal of this library is to generate nonlinear inverse models for the controller. The library contains state-of-the-art Low Earth Orbit (LEO) space environment models and components. It was implemented in the Modelica modeling language [23].

The theory of satellite dynamics is well understood and many important aspects of spacecraft modeling

are described in publications such as [24]. A wide range of satellite simulators exist [3, 2, 11, 5, 36] that are able to accurately simulate a satellite in orbit.

There also exist simulators based on the Modelica modeling language. The authors made use of the advantages of using the Modelica language for the implementation and the inversion of rigid satellites [29]. In addition, several related publications such as [30, 20, 19] describe the simulation of satellites in detail. Satellites with flexible appendages were considered in [33]. Many important aspects of spacecraft modeling are already covered in these publications and promising results were reported by these authors.

The Space Systems Library was developed to implement nonlinear inverse models of satellites with flexible structures, such as solar panels, that cannot be easily implemented within existing simulators. Having direct access to all component equations from the SSL allows the successful implementation of algorithms from recent results obtained in the field of robotics regarding the inversion of flexible multi-body systems and trajectory optimization based on inverse models [32, 31].

The new DLR internal library builds upon the Modelica Standard Library [23], and especially the Modelica MultiBody Library [25], as well as the DLR FlexibleBodies Library [12], DLR Visualization Library [1] and the DLR Optimization Library [27]. The SSL combines their capabilities to achieve a wide range of possible applications, ranging from visualization of space missions to high accuracy simulations, optimizations and development of control systems for satellites with flexible structures.

The objective of this paper, apart from introducing the SSL, is to demonstrate its capabilities by modeling a near Earth satellite based on the TET-1 prototype [8], which is part of the FireBird mission – a mission of the German Aerospace Center (DLR) for fire reconnaissance [4, 16]. The modeling of the satellite is described in section 3.

The satellite is modeled with flexible solar panels<sup>1</sup>. From the nonlinear (direct) satellite model, an inverse model is derived (sec. 4). This inverse model is used to calculate optimal trajectories for a reorientation maneuver of the satellite using the reaction wheels under constraints (sec. 5) and can also be used as a feed-forward controller.

## 2 The Space Systems Library

The SSL enables object-oriented, acausal, and equation-based modeling of space systems dynamics and its corresponding environments. This in turn allows controller design and verification, as well as development of path planning and other algorithms.

**The Space Systems world model** The default world model of the Modelica MultiBody Library [25] is exchanged by a new world model which is compatible with the default world model. It offers additional options and methods for space environment simulations. The world model handles the global simulation time that is used to calculate planet positions and various transformations. The initial time can be given in calendar- or Julian date format. The latter is also implemented internally as time format. The basis coordinate system is chosen to be the Earth Centered Inertial (ECI) coordinate system, which is suitable for near earth satellite simulations. The world model offers a connector and transformation for the Earth Centered Earth Fixed (ECEF) coordinate system, which is useful for the simulation of objects on earth, like emitter stations. The transformation from ECI to ECEF coordinate system is computed as described in [24, 15]. This calculation also considers the difference in seconds between Universal and Universal Coordinated Time (leap seconds,  $t_{UT1-UTC}$ ) that has to be given as an initial value and can be taken from tabular data [37].

**Gravity acceleration computation** The gravity acceleration  $g_0 \in \mathbb{R}^3$  plays a very important role for the simulation of satellites. Hence, multiple gravity models of different complexity were implemented. The most precise model implemented is the EGM96 gravity model [18]. A computational efficient approximation of this model was implemented, which uses terms of up to the second degree of the zonal harmonic coefficients of the gravitational potential. In addition,

<sup>1</sup>The term "flexible satellite" will be used hereafter as a shorthand notation.

moon and sun gravities have been included, considering them as important perturbation factors. These are modeled as point gravity [24]. Although newer, more advanced gravity models exist, the accuracy of the chosen models is sufficient for our multi-body approach that focuses on short term simulations.

**Gravity gradient torque** The gravity gradient torque is modeled as a torque  $\tau_a$  that acts on the frame  $a$  (position  $r_{0,a} \in \mathbb{R}^3$  and orientation  $R_a \in \mathbb{R}^{3 \times 3}$ . The index  $a$  is used to describe a generic frame, which is instantiated for every object) to which it is connected. This frame should be connected to the center of mass of the body in consideration. The torque is caused by the mass distribution of the body in consideration, and depends on the inertia tensor  $I \in \mathbb{R}^{3 \times 3}$  as follows.

$$\tau_a = \left( R_a \cdot g_0(r_{0,a}, t_J) \frac{3}{\|r_{0,a}\|} \right) \times \left( I \cdot R_a \frac{-r_{0,a}}{\|r_{0,a}\|} \right) \quad (1)$$

In eq. (1), the gravity acceleration vector  $g_0 \in \mathbb{R}^3$  is a function of the position  $r_{0,a}$  and the Julian date  $t_J$  (to compute planet positions). See [17] for more details.

**Solar radiation pressure** The effect of the solar radiation pressure  $p_\odot$  is modeled as a force element  $f_{sp} \in \mathbb{R}^3$  that acts on the element to which it is connected. Shadows of the moon and sun are considered using a cylindrical shadow model. The models are implemented as proposed in [24].

**Atmospheric drag** The atmospheric drag is caused by friction with the atmosphere depending on the height of the satellite above the Earth. Like the radiation pressure, it is modeled as a force and torque element acting on the attached body which should be located at the center of pressure.

The density  $\rho(\lambda, \phi, h, t_J)$  of the atmosphere is computed using the NRLMSISE-00 atmospheric density model [28]. The density  $\rho$  depends on the longitude  $\lambda$ , latitude  $\phi$  and height above earth  $h$ , that can be computed from  $r_{0,a}$ , as well as the actual Julian date  $t_J$ . The drag force  $f_a \in \mathbb{R}^3$  and torque  $\tau_a \in \mathbb{R}^3$  can be computed using eq. (2).

$$v_{rel} = \dot{r}_{0,a} - \omega_\oplus \times r_{0,a} \quad (2a)$$

$$f_a = -R_a \frac{1}{2} c_d A_{ad} \rho \|v_{rel}\| v_{rel} \quad (2b)$$

assuming  $\|v_{rel}\| \neq 0$  (otherwise  $\tau_a = 0$ ):

$$\tau_a = R_a \frac{1}{2} c_d A_{ad} \rho \|v_{rel}\|^2 \left( \frac{v_{rel}}{\|v_{rel}\|} \times (R_a s_{cp}) \right) \quad (2c)$$

In eq. (2),  $\omega_{\oplus} \in \mathbb{R}^3$  is the earth angular velocity,  $c_d$  is the drag coefficient,  $A_{ad}$  is the affected area (which can depend on  $R_a$ ) and  $s_{cp} \in \mathbb{R}^3$  is the vector from the center of pressure to the center of mass, all resolved in the attached frame  $a$ .

**Geomagnetic field** The geomagnetic field can be computed at a desired frame (position  $r_{0,a}$ ) by a geomagnetic field component, using the US/UK World Magnetic Model [22]. The model provides a magnetic field vector  $B_m(\lambda, \phi, h, t_J) \in \mathbb{R}^3$  that depends on the longitude  $\lambda$ , latitude  $\phi$  and height above earth  $h$ , which can be computed from  $r_{0,a}$  and the Julian date  $t_J$ .

**Variable mass systems** The effects of variable mass systems are very important for the modeling of launchers but their consideration can also be necessary for satellites that use gas thrusters, if high precision for the simulation is required. If mass flow  $\dot{m} \in \mathbb{R}$  is small, variable mass is usually neglected. Effects of thrusters and jets together with their tanks can be modeled as variable mass systems.

For the SSL, variable mass systems were implemented based on [9, 10] using the concept of a variable mass cylinder with different models of fuel burning. The cylinder represents the fuel or gas tank that is directly attached to the nozzle (of the thruster or jet).

### 3 The satellite model with flexible structures

The satellite model consists of flexible structures that are modeled as modal bodies as described in [12]. They are based on the definition of Standard Input Data (SID) as defined in [38] as well as rigid bodies and powertrain elements. The equations of motion of each flexible part  $i$  are given in eq. (3). The  $\tilde{(\cdot)}$  operator is used to generate a skew-symmetric matrix of a vector.

$$\begin{pmatrix} m_i I_3 & & \text{sym.} \\ m_i \tilde{d}_{CM,i}(q_i) & \Theta_i(q_i) & \\ C_{t,i}(q_i) & C_{r,i}(q_i) & M_{e,i} \end{pmatrix} \begin{pmatrix} a_{R,i} \\ \dot{\omega}_{R,i} \\ \dot{q}_i \end{pmatrix} + \begin{pmatrix} 2\tilde{\omega}_{R,i} C_{t,i}^T(q_i) \dot{q}_i + \tilde{\omega}_{R,i} \tilde{\omega}_{R,i} d_{CM,i}(q_i) \\ G_{r,i}(\dot{q}_i) \tilde{\omega}_{R,i} + \tilde{\omega}_{R,i} \Theta_i(q_i) \omega_{R,i} \\ G_{e,i}(\dot{q}_i) \tilde{\omega}_{R,i} + O_{e,i}(q_i) \Omega(\omega_{R,i}) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ K_{e,i} q_i + D_{e,i} \dot{q}_i \end{pmatrix} = \begin{pmatrix} h_{t,i} \\ h_{r,i} \\ h_{e,i} \end{pmatrix} \quad (3)$$

The meaning of the symbols in eq. (3) are listed in table 1. The file of a flexible body (SID) can be ob-

Table 1: SID abbreviations

$a_R, \omega_R \in \mathbb{R}^3$	acceleration of the ref. frame
$q \in \mathbb{R}^{n_e}$	modal amplitudes
$m \in \mathbb{R}$	body mass
$I_3 \in \mathbb{R}^{3 \times 3}$	identity matrix
$d_{CM}(q) \in \mathbb{R}^3$	position of center of mass
$\Theta(q) \in \mathbb{R}^{3 \times 3}$	inertia tensor
$C_t(q) \in \mathbb{R}^{n_e \times 3}$	inertia coupling mat. (trans.)
$C_r(q) \in \mathbb{R}^{n_e \times 3}$	inertia coupling mat. (rot.)
$k_{\omega}(\omega_R, q, \dot{q}) \in \mathbb{R}^{3+n_e}$	gyro. and centrifugal forces
$\Omega(\omega_R) \in \mathbb{R}^6$	components of $\omega_R$
$G_e(\dot{q}) \in \mathbb{R}^{n_e \times 3}$	gyroscopic matrix (modal)
$G_r(\dot{q}) \in \mathbb{R}^{n_e \times 3}$	gyroscopic matrix (rot.)
$O_e(q) \in \mathbb{R}^{n_e \times 6}$	centrifugal matrix (modal)
$h(q) \in \mathbb{R}^{3+n_e}$	external forces
$M_e \in \mathbb{R}^{n_e \times n_e}$	modal mass matrix
$K_e \in \mathbb{R}^{n_e \times n_e}$	modal stiffness matrix
$D_e \in \mathbb{R}^{n_e \times n_e}$	modal damping matrix

tained from an FEM-analysis (e.g. using ANSYS<sup>®</sup> or ABAQUS<sup>®</sup> in combination with SIMPACK<sup>®</sup>) directly from CAD and material data of the component. In the modal reduction,  $n_e$  modes are selected and the required data to calculate eq. (3) is stored in the SID file. The flexible bodies can be combined with other flexible and rigid bodies to model the structural dynamics of the satellite.

The reaction wheels are driven by a motor together with a powertrain including friction. Furthermore, most powertrains used in today's satellites are very stiff. Future satellite designs may incorporate more lightweight constructions with elastic effects in the powertrains, in combination with more powerful and agile motors, as in robotics today. For this reason nonlinear elasticity between the motor and the reaction wheel disk can be taken into account. The elasticity can result from the material of the drive shaft or due to the construction of the coupling. By combining one dimensional models of the powertrain with three dimensional inertia and mass elements, computational efficient models can be designed by using mounting and rotor elements as described in [34]. To model a powertrain in a way which can also be used to generate inverse models is described in [32, 31]. It consists of approximate friction and nonlinear elasticity models that have strictly monotonic characteristics.

Using the components of the library, a detailed mechanical satellite model can be built. The benchmark model used here is a satellite model which is based

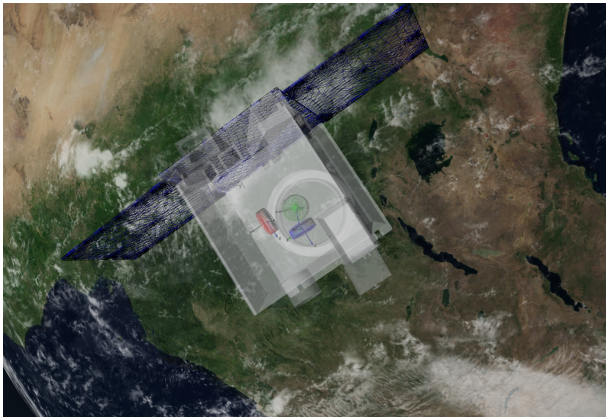


Figure 1: Animation of the satellite with the SSL.

on the TET-1 prototype [8], with slightly exaggerated elasticities, in order to demonstrate the capabilities of the library and future lightweight satellite designs. Three solar panel elements have been modeled as modal bodies. These have been generated from CAD data, a subsequent FEM-analysis, followed by a modal reduction and finally obtaining an SID model. The main body of the satellite is modeled as a rigid body because it is considerably less flexible than the solar panels. Fig. 1 shows an animation of the satellite generated with the SSL. Three reaction wheels mounted at the main satellite body are modeled as rigid bodies representing flywheels. They are connected to one-dimensional (rotational) flexible powertrains with friction and first order motor dynamics. The gyroscopic torques of the one-dimensional powertrain elements are considered in the model by using the mounted rotor elements as described in [34]. The gravity  $g_0$  which is acting on the rigid and flexible bodies is calculated using the gravity model, which is defined by the (global) satellite world model. Additional force and torque elements are connected to the satellite to account for the solar radiation pressure and atmospheric drag as well as the gravity gradient torque, described in sec. 2.

#### 4 Inversion of a structural elastic satellite model

On the top level, a satellite dynamics model has the structure as shown in the top half of fig. 2.

The motor currents  $I_m$  are used to drive the reaction wheels of the satellite, from which in turn the rotation of the satellite is determined by the solution of a differential-algebraic equation system. In particular, the angular velocity  $\omega_{ACS}$  of the central satellite frame

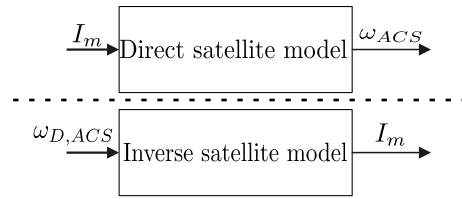


Figure 2: Top level view of direct and inverse satellite model.

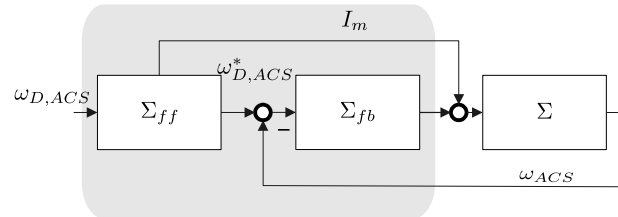


Figure 3: Two degree of freedom control of the satellite motors. With feed-forward controller  $\Sigma_{ff}$ , feedback controller  $\Sigma_{fb}$  and satellite  $\Sigma$ .

is computed, it will be referred to as the ACS-frame (Attitude Control System). This computation is a standard task of satellite simulators.

For advanced control systems, the inverse of this model is needed, as shown in the lower half of fig. 2. The previously given motor currents  $I_m$  shall now be computed from the desired angular velocity  $\omega_{D,ACS}$ . The nonlinear inverse model is based on the satellite model described in section 3. The basis for the inversion of a nonlinear (Modelica) model are index-reduction techniques using the algorithm of Pantelides. The index reduction method [26, 21] allows to choose which equations have to be used to generate a DAE of (at most) index one. All Modelica simulators support this or similar algorithms and automatically perform the inversion. However, both the generation of the inverse model, as well as the numerical computation may fail, if the underlying model does not fulfill certain requirements. Especially, the system must be smoothly continuously differentiable up to the necessary order of differentiation that is determined by the algorithm of Pantelides.

Such a nonlinear inverse model can be, for example, used as feed-forward controller  $\Sigma_{ff}$  for the inner control loop, see fig. 3. An additional outer feedback control loop based on the satellites star tracker would be needed to achieve stationary accuracy of the satellite. For a rigid satellite with rigid powertrains without motor dynamics  $\omega_{D,ACS}$  and the equations of motion have to be continuous differentiable at least once so that the equations for the inverse model can be solved via the

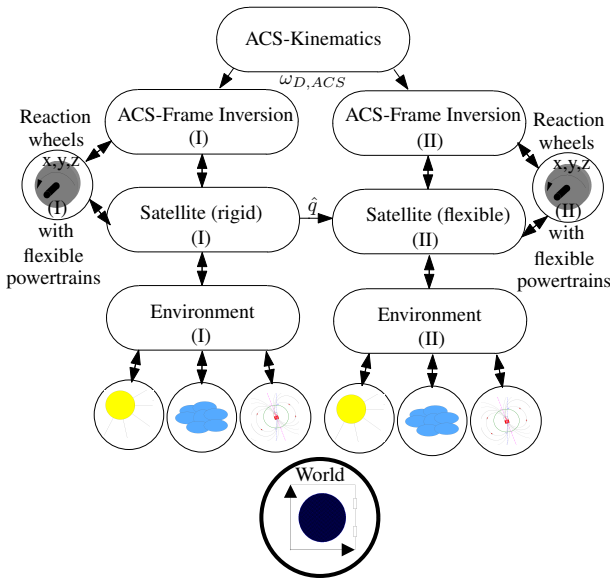


Figure 4: Setup of the approximate inverse satellite model with flexible solar panels and powertrains. The global world model provides the gravity model and simulation time. The environment causes forces and torques on the satellite as described in section 2. Because the environment effects can depend on the states of the satellite, they have to be included for both parallel models of the satellite (I & II).

algorithm of Pantelides. For a satellite with flexible structures and flexible powertrains and first order motor dynamics, as described in section 3,  $\omega_{D,ACS}$  as well as the equations of motion have to be differentiable at least three times. In practice, this should be increased by at least one order to avoid non-differentiable motor currents. If these higher order derivatives of  $\omega_{D,ACS}$  are not available, a low-pass filter can be used to approximately compute  $\omega_{D,ACS}^*$  and its derivatives (see [35, 32, 31]). This is a reason why exact sticking friction cannot be used as part of the inverse model and an approximation has to be used. For components that use non differentiable (e. g. tabular) data like the NRLMSISE-00 atmospheric density model [28] the derivatives have to be either approximated or neglected. This can be achieved in Modelica by using user-defined derivative-functions instead of automatic symbolic differentiation.

It is also necessary that the inverse model is stable. For a linear model this means that the model to be inverted has no (transmission-) zeros in the right half of the complex plane. For the nonlinear model this means that the zero dynamics has to be stable. For highly nonlinear models it is usually very difficult or even impossible to calculate the zero dynam-

ics analytically (e. g. using methods described in [13] that involve solving partial differential equations). Often, the only practical way is to calculate linearizations of the nonlinear model to verify the stability of the (transmission-) zeros.

One typical reason for unstable (transmission-) zeros for systems with flexible structures is the non-collocation of inputs (actuators) and outputs (sensors) of the system. For structural elastic robots this can be the case if motor torques are chosen as output and the robot tip position is chosen as input of an inverse model [32]. The same problem is also possible for flexible satellites, e. g. if the main satellite body attitude should be controlled by actuators that are positioned on a flexible mounting. In general, in order to achieve a stable inverse model of a flexible satellite, approximations must be used, as explained in sec. 4.1, because the exact inverse model is unstable. A first step for the construction of the satellite's inverse model is the definition of a base ACS-frame. The desired angular velocity  $\omega_{D,ACS}$  of the satellite will be given with respect to this frame. A good choice for the ACS-frame is either the center of mass of the satellite, or the tip-position of an important on-board instrument of the satellite, e. g. a mounted camera or sensor. This frame is imposed as a root frame according to the Modelica MultiBody Library definition [25]. Hence, the orientation  $R_a \in \mathbb{R}^{3 \times 3}$  as well as the angular velocity  $\omega_a$  of the frame are computed consistently. In addition, the kinematic tree of the satellite is rooted in this frame, although the frame itself can be moving. This is necessary to handle overdetermined DAEs with symbolic transformation techniques in a way that is used in the Modelica MultiBody Library (for further details see the appendix in [25]). The translation of the ACS-frame is not restricted and it moves according to the external forces and the gravity, which act on the bodies connected to it. This results in a hybrid forward/inverse model corresponding to the translation and orientation. The model inversion is done by using the desired angular velocity  $\omega_{D,ACS}$  as input for the local angular velocity of the ACS-frame  $\omega_a$ , starting from a defined angular velocity  $\omega_{a,0}$  and orientation. The original input for the forward model  $I_m$ , which is the current of the three reaction wheels, is chosen as the new output for the inverse model. The computation of the required angular velocity  $\omega_{D,ACS}$ , for a specific maneuver, will be described in section 5.

The ACS-frame is connected to the satellite body with its actuators. The desired angular velocity  $\omega_{D,ACS}$  of the ACS-frame is achieved by adding a set of equa-

tions to force the angular velocities to the desired values (using Modelica inverse block constraints). This results in three additional equations for the calculation of the torques of the attached torque wheels  $\tau_w \in \mathbb{R}^3$ . The resulting equations for the ACS-frame model are given in eq. (4).

$$R_a = f_{rot}(\phi_{ACS}) \quad (4a)$$

$$\omega_a = f_{\omega}(\phi_{ACS}, \dot{\phi}_{ACS}) = \omega_{D,ACS} \quad (4b)$$

$$\tau_a = -\tau_w \quad (4c)$$

In eq. (4),  $f_{rot}$  and  $f_{\omega}$  are functions to calculate the transformation matrix  $R_a$  and local angular velocity  $\omega_a$ . Although the values of the Cardan angles  $\phi_{ACS}$  can jump, the internal representation of the orientation is calculated using rotation matrices that remain continuous and show no singularities (see [25] for implementation details). The ACS-frame is forced to move along the desired orientation and the required torques are generated by the attached reaction wheels via  $\tau_w$  from which the motor currents  $I_m$  can be computed by inversion of the elastic powertrain and first order motor dynamics [32, 31]. This is only possible if the reaction wheels are mounted in such a way that they can generate the required torque vector  $\tau_w$ . Otherwise the equation system will not be solvable (singular). If more than three reaction wheels are used, a torque allocation algorithm has to be implemented in addition.

#### 4.1 Approximation of the deformation of flexible structures of an inverse satellite model

For satellites with flexible structures, depending on the location of the ACS-frame, the exact inverse system can be unstable, and therefore not useful for a control system. This can be checked by calculating the transmission-zeros of the linearized system. If they have a positive real part, the inverse model will be unstable. They can be caused by the combination of flexible structures and the chosen ACS-frame location. In addition, the exact inversion of the equations of motion of flexible structures with weak damping can lead to numerical instability and stiff systems. A solution for this problem is to obtain an approximation of the elastic deformations for the inverse satellite model. Our method is based on a quasi-static approach using two parallel models. A similar method was already successfully implemented for the inversion of flexible robot arms in [32, 31]. Starting points are flexible bodies modeled after eq. (3) from an SID file. In the first

model, quasi-static approximations for the elastic deformation  $\hat{q} \in \mathbb{R}^{n_e}$  are calculated. In the second model, these deformations are used as input for the flexible parts and the resulting forces are re-calculated. Figure 4 shows an overview for the setup of the inverse satellite model with flexible solar panels and powertrains. For the first of the two parallel models, the equations of motion for a flexible body are given in eq. (5). The index  $i$  for each component is omitted here.

$$\begin{pmatrix} mI_3 & sym. \\ m\tilde{d}_{CM}|_{q=0} & \Theta|_{q=0} \end{pmatrix} \begin{pmatrix} a_R^{(r)} \\ \dot{\omega}_R^{(r)} \end{pmatrix} \quad (5a)$$

$$+ \begin{pmatrix} \tilde{\omega}_R^{(r)} \tilde{\omega}_R^{(r)} d_{CM}|_{q=0} \\ \tilde{\omega}_R^{(r)} + \tilde{\omega}_R^{(r)} \Theta|_{q=0} \omega_R^{(r)} \end{pmatrix} = \begin{pmatrix} h_r^{(r)} \\ h_r^{(r)} \end{pmatrix} \quad (5b)$$

$$h_e^{(r)} = C_t|_{q=\hat{q}} g_{ref}^{(r)} \quad (5b)$$

$$+ \sum_j^{n_{node}} \left( \Phi_{m,j}^T|_{q=\hat{q}} f_{ref,j}^{(r)} + \Psi_{m,j}^T|_{q=\hat{q}} \tau_{ref,j}^{(r)} \right)$$

$$= C_t|_{q=\hat{q}} a_R^{(r)} + C_r|_{q=\hat{q}} \dot{\omega}_R^{(r)} + G_e|_{q=\hat{q}} \omega_R^{(r)}$$

$$+ O_e|_{q=\hat{q}} \Omega(\omega_R^{(r)}) + K_e \hat{q} + D_e \dot{\hat{q}}$$

The notation  $(\cdot)^{(r)}$  denotes variables calculated using the assumption that forces which cause the deformations can be approximated by neglecting elastic deformations (quasi-static). This also leads to a simplification of terms that would normally depend on the elastic deformation  $q$ . Therefore, the flexible structural parts are calculated as rigid bodies and approximations for the deformations  $\hat{q}$  are calculated using eq. (5) in the first parallel model. To get a stable approximation, terms for the second derivative with respect to time involving  $\ddot{\hat{q}}$  are neglected in eq. (5), so that a nonlinear first order differential equation (that is linear in its highest derivative  $\dot{\hat{q}}$ ) is obtained to approximate  $q$  in the inverse model.

The approximation  $\hat{q}$  is used as input for the second (parallel) inverse satellite model. In the second model, the approximations for  $\hat{q}$  are used to recalculate all forces and positions. This is necessary because the resulting deformations change the forces and torques that act on the connected bodies. The equations of motion of the flexible parts in the second model are given

in eq. (6).

$$\begin{pmatrix} mL_3 & m\tilde{d}_{CM}^T|_{q=\hat{q}} & C_t^T|_{q=\hat{q}} \\ m\tilde{d}_{CM}|_{q=\hat{q}} & \Theta|_{q=\hat{q}} & C_r^T|_{q=\hat{q}} \end{pmatrix} \begin{pmatrix} a_R^{(II)} \\ \dot{\omega}_R^{(II)} \\ \ddot{\hat{q}} \end{pmatrix} + \begin{pmatrix} 2\tilde{\omega}_R^{(II)}C_t^T|_{q=\hat{q}}\dot{\hat{q}} + \tilde{\omega}_R^{(II)}\tilde{\omega}_R^{(II)}d_{CM}|_{q=\hat{q}} \\ G_{rn}|_{q=\hat{q}}\tilde{\omega}_R^{(II)} + \tilde{\omega}_R^{(II)}\Theta|_{q=\hat{q}}\omega_R^{(II)} \end{pmatrix} = \begin{pmatrix} h_t^{(II)} \\ h_r^{(II)} \end{pmatrix} \quad (6)$$

Because the approximation of the modal amplitudes  $\hat{q}$  are used directly as input, equations for the modal forces  $h_e$  are not necessary in eq. (6). In this equation, the second derivative  $\ddot{\hat{q}}$  is again considered. In general, if the flexible parts are connected to each other, it is also possible that the reference accelerations and velocities as well as the external forces change, so they are denoted with the notation  $(\cdot)^{(II)}$ .

This approximation is valid for small deformations and smooth reorientation maneuvers without extreme external forces and can always be calculated where otherwise no solution could be found (because the exact inverse model would be unstable, depending on the chosen ACS-frame location). If the approximation is sufficient for a specific application can be verified by simulating the approximate inverse model in combination with the original forward model and looking at the resulting error in the calculated torques and deformations. If the resulting error is within the mission tolerance, the approximation can be used. A feedback controller can minimize the remaining error. A typical result for the approximation shows fig. 5, where the first bending mode of an outer solar panel is compared to the resulting first mode of the solar panel without approximation, when using the computed motor currents  $I_m$  as feed-forward command (without any feedback controller). The approximation  $\hat{q}_1$  follows  $q_1$  closely, but a residual vibration remains at the end of the movement which results from the (stable) first order approximation for the modal amplitude in eq. (5). This vibration can be further damped by a feedback-controller.

The resulting state  $x_{inv}$ , input  $u_{inv}$  and the output  $y_{inv}$  for the inverse model setup of fig. 4 is given in eq. (7). The state of the inverse model consists of the states of several subsystems. The states of the ACS-frame  $x_{ACS}$ , the three powertrains  $x_{PT}$  and the states of the approximation of the modal amplitudes  $\hat{q}_s$  for the three solar panel elements. They are modeled with  $n_e = 3$  modes for each panel. For the powertrains the flexibility of the connection to the reaction wheels is modeled as spring damper systems (states

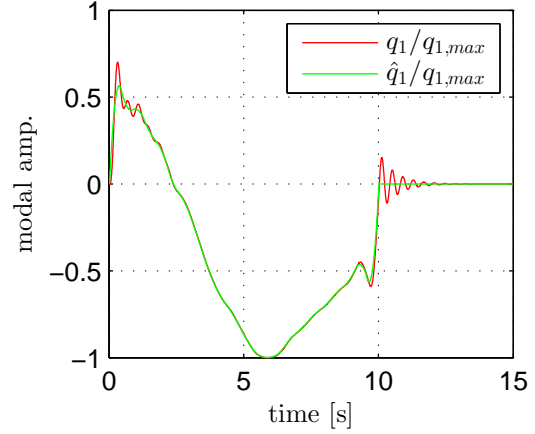


Figure 5: Comparison of the first mode (normalized) between approximation  $\hat{q}_1$  and resulting mode  $q_1$ .

$\phi_{rel,PT}, \dot{\phi}_{rel,PT}$ ). Because the parallel model setup is used, two sets of states (I & II) are needed for  $x_{ACS}$  and  $x_{PT}$ .

$$x_{inv} = \left( x_{ACS}^{(I)}, x_{ACS}^{(II)}, x_{PT}^{(I)}, x_{PT}^{(II)}, \hat{q}_s \right)^T \in \mathbb{R}^{51} \quad (7a)$$

$$u_{inv} = \omega_{D,ACS} \in \mathbb{R}^3, y_{inv} = I_m \in \mathbb{R}^3 \quad (7b)$$

with:

$$x_{ACS} = (\phi_{ACS}, r_{0,ACS}, v_{0,ACS}) \quad (7c)$$

$$x_{PT} = (\phi_m, \dot{\phi}_m, \phi_{rel,PT}, \dot{\phi}_{rel,PT}) \quad (7d)$$

If the input for  $u_{inv}$  is not differentiable, a filter has to be used and the states of the filter are then also part of  $x_{inv}$ . Simulation results showed that the inverse satellite model can be used to accurately calculate the required motor currents and estimate the deformations of the flexible parts using the new approximation method for typical trajectories.

## 5 Computing optimal trajectories using inverse satellite models

The goal of the attitude trajectory optimization is to find optimal motor currents  $I_m \in \mathbb{R}^3$  reorienting the satellite from a starting orientation  $Q_{D,0}$  (using a unit quaternion representation  $Q_D \in \mathbb{R}^4$ ) at time  $t_0$  to a desired end orientation  $Q_{D,f}$  at time  $t_f$ . Using a forward model of the satellite with flexible structures, this op-

timization problem is defined by eq. (8).

$$\min \left\{ \xi_t t_f + \xi_I \sum_{i=1}^3 \left( \int_{t_0}^{t_f} |I_{m,i} \dot{\phi}_{m,i}| dt \right) \right\} \quad (8a)$$

with:

$$F(t, x, \dot{x}, z, u) = 0, x(t_0) = x_0, t \in [t_0, t_f] \quad (8b)$$

$$Q(x_0, z_0, t_0) = Q_{D,0} \quad (8c)$$

constrained by:

$$|\dot{\phi}_m| \leq \dot{\phi}_m^{max} \quad (8d)$$

$$Q(x, z, t) = Q_{D,f} \text{ for } t \geq t_f \quad (8e)$$

with control function:

$$u_c(t) = I_m(t) \in \mathbb{R}^3, |I_{m,i}| \leq I_m^{max} \quad (8f)$$

The chosen criteria allows the optimization of time- or energy-optimal attitude trajectories (or mixtures). The term  $F(t, x, \dot{x}, z, u) = 0$  represents the nonlinear satellite model in implicit DAE form where  $t$  denotes the time,  $x$  the states,  $u$  the inputs (unknown motor currents  $u = I_m$ ) and  $z$  the algebraic variables of the system. The scalar factors  $\xi_t, \xi_I \in \mathbb{R}_+$  are used to weight the criteria for the optimization. The factor  $\xi_t$  weights the resulting end time  $t_f$  for which the end orientation  $Q_{D,f}$  is reached, while  $\xi_I$  is a weight for the energy criteria and  $\dot{\phi}_m \in \mathbb{R}^3$  are the motor angular velocities for the three motors.

The optimization problem in eq. (8) is difficult to solve directly because of the large solution space for  $I_m(t)$  in combination with the strict constraint for the end orientation. By using an inverse satellite model with flexible structures from section 4 and a parameterization of the orientation, the optimization problem can be greatly simplified. The desired orientation is restricted to a path  $Q_D(s(t))$  which is calculated by using a spherical linear quaternion interpolation (SLERP, see [6]) along a scalar path parameter  $s(t) \in [0, 1]$ . This results in an interpolation from the starting orientation  $Q_{D,0} = Q_D(s=0)$  to the end orientation  $Q_{D,f} = Q_D(s=1)$ . The desired angular velocity  $\omega_{D,ACS}$  for the ACS-frame which is used as input for the inverse satellite model can be calculated using eq. (9) where  $\omega_0$  is the initial angular velocity of the ACS-frame resolved in the ECI-frame and  $q_i$  are scalar elements of  $Q_D$ .

$$\omega_{D,ACS} = 2 \begin{pmatrix} q_4 & q_3 & -q_2 & -q_1 \\ -q_3 & q_4 & q_1 & -q_2 \\ q_2 & -q_1 & q_4 & -q_3 \end{pmatrix} \dot{Q}_D \quad (9)$$

$$+ 2((q_4 q_4 - 0.5)\omega_0 + ((q_1, q_2, q_3)^T \omega_0)(q_1, q_2, q_3)^T - q_4((q_1, q_2, q_3)^T \times \omega_0))$$

The resulting optimization problem is described in

eq. (10).

$$\min \left\{ \xi_t t_f + \xi_I \sum_{i=1}^3 \left( \int_{t_0}^{t_f} |I_{m,i} \dot{\phi}_{m,i}| dt \right) \right\} \quad (10a)$$

with:

$$F(t, x, \dot{x}, z, u_c) = 0, x(t_0) = x_0, t \in [t_0, t_f] \quad (10b)$$

constrained by:

$$|\dot{\phi}_m| \leq \dot{\phi}_m^{max} \quad (10c)$$

$$|I_{m,i}| \leq I_m^{max} \quad (10d)$$

$$\int_0^{t_f} u_c(t) dt = s(t_f) = 1 \quad (10e)$$

with control function:

$$u_c(t) = \dot{s}(t) \in \mathbb{R}_+ \quad (10f)$$

$$u_c(t_0) = u_c(t_f) = \dot{u}_c(t_0) = \dot{u}_c(t_f) = 0 \quad (10g)$$

To achieve a finite optimization problem, the infinite possibilities for the path parameter  $s(t)$ , given by the integration over the control function  $u_c(t)$ , have to be limited by using an appropriate parameterization. This is performed by using a B-spline [7] of order  $n_s = 3$  to parameterize  $u_c(t)$ . Using inverse models in combination with a path parameter  $s(t) \in [0, 1]$  inside the optimization offers great advantages over a direct optimization of the motor currents:

- The number of necessary tuners<sup>2</sup> is much smaller. Instead of having to parametrize all three motor currents  $I_m(t)$ , only one scalar function  $u_c(t)$  has to be parametrized.
- The optimization does not have to stabilize the system like a controller. Using the desired path  $Q_D(s)$  as input for the inverse model results in reaching  $Q_{D,f}$  exactly.

The equality condition  $s(t_f) = 1$  can still be difficult to achieve for optimization algorithms. There are two possible ways to overcome this. One possibility is to avoid the equality condition by using an additional criteria for the optimization in the form of eq. (11).

$$(s(t_f) - 1)^2 \rightarrow \min \quad (11)$$

However, this can lead to an unacceptable error in the end-orientation  $Q_{D,f}$ . A better way to avoid the equality condition is to directly reshape the B-spline control vector  $d_c$  each time the tuner vector is modified by the optimization algorithm. The modified vector is called

<sup>2</sup>The word tuners is used for the free parameters of an optimization that are changed by the optimizer.



$d_c^+ \in \mathbb{R}^{n_c}$  and is calculated using eq. (12).

$$d_c^+ = \frac{d_c}{\int_0^{t_f} \left( \sum_{i=1}^{n_c} d_{c,i} N_{i,n_s}(t) \right) dt} \quad (12a)$$

$$\Rightarrow s(t_f) = \int_0^{t_f} \left( \sum_{i=1}^{n_c} d_{c,i}^+ N_{i,n_s}(t) \right) dt = 1 \quad (12b)$$

This reshaping leads directly to  $s(t_f) = 1$  so that the optimizer does not have to fulfill the equality condition by itself. For time optimal optimizations  $t_f$  is also a tuner and the discrete B-spline control points  $t_i$  are reshaped to lie in the interval  $[0, t_f]$  if  $t_f$  is modified by the optimization algorithm.

## 5.1 Trajectory optimization results

Trajectory optimizations on the basis of an inverse satellite model were performed for a reorientation maneuver of  $28^\circ$  around the axis  $e_{rot} = (1, 1, 1)^T$ .

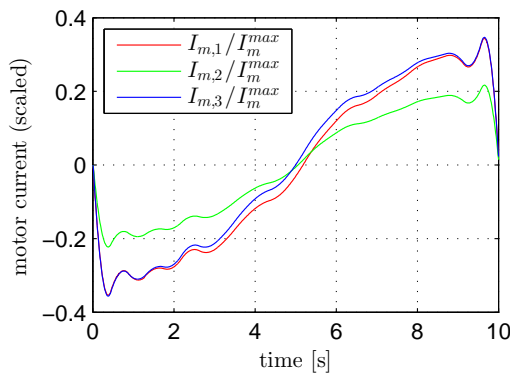


Figure 6: Motor currents for the energy optimal control function.

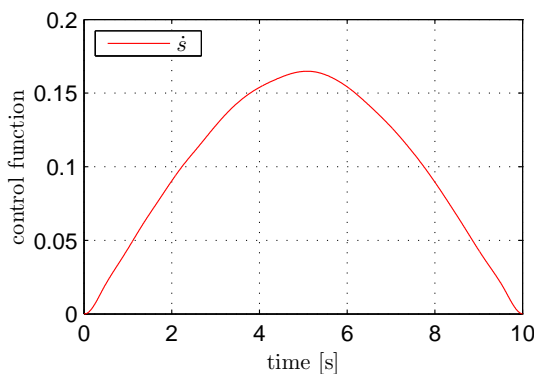


Figure 7: Energy optimal control function for fixed  $t_f$ .

All optimization are performed using a B-spline parameterization of the control function with  $d_c \in \mathbb{R}^{30}$  as described in the last section. Starting values are found

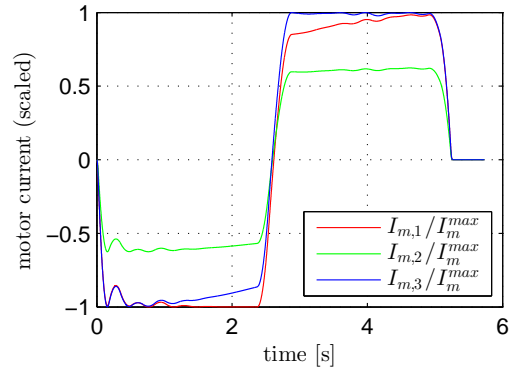


Figure 8: Motor currents for the time optimal control function.

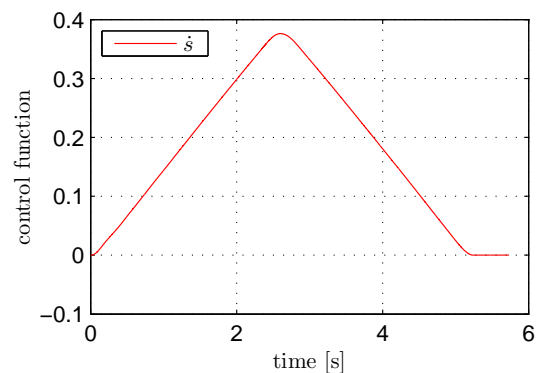


Figure 9: Time optimal control function.

using a global genetic algorithm with a population size of 100 and a rigid inverse satellite model. After good initial parameters have been found, the parameters are refined using a local pattern search algorithm [14, 27] with a complete approximate structural elastic inverse satellite model as described in sec. 3 (including flexible powertrains and flexible solar panels). For both optimization steps the reshaping of the tuners was used as described in the last section.

The first optimization was performed for a fixed end-time  $t_f = 10s$ . It was the goal to minimize the required energy of this maneuver, according to the criteria given in eq. (10) with  $\xi_t = 0$  and  $\xi_I = 1$ . The plots in fig. 6 and fig. 8 show the resulting optimized motor currents and the corresponding control functions as shown in fig. 9 and fig. 7. The second optimization was performed for the parameters  $\xi_t = 1$  and  $\xi_I = 0$  in eq. (10) with free end time  $t_f$  (as tuner) and results in a time optimal solution for the given path. The resulting trajectories allow to reorient the satellite according to the desired SLERP path. Although there are still oscillations of the flexible components (e. g. solar panels), they are compensated by the reaction wheels

such that the ACS-frame follows the desired path. In addition, limitations on the motor currents and motor velocities are maintained. Small errors that result from the approximations used for the generation of the inverse model can be compensated by a feedback controller (in addition to modeling errors in the case of a real satellite).

## 6 Conclusion

In this paper, we have presented the Space Systems Library which provides components for satellite/spacecraft nonlinear modeling in Low Earth environment. We demonstrated the library capabilities with a new method for model-based attitude control of a satellite with flexible solar panels.

The method consists on transferring the inverse flexible model approach, successfully implemented in the field of industrial robotics, to the application of flexible satellites modeling and control.

The inverse satellite model allows the computation of the motor currents for a given trajectory of the satellite in a way that the elasticity of the powertrains and flexible solar panels is taken into account and compensated. Using the inverse model, optimal re-orientation maneuvers have been computed for energy and time optimality. The use of inverse satellite models in this optimization greatly improves the optimization process by simplifying the problem considerably.

As part of a two degree of freedom control system, the inverse satellite model can be used as a feed-forward controller to compensate elastic effects and other modeled nonlinear effects that act on the satellite.

## 7 Acknowledgment

The helpful remarks of M. Otter and A. Pfeiffer for this work are gratefully appreciated.

## References

- [1] T. Bellmann. Interactive simulations and advanced visualization with modelica. *Proceedings of the 7th Modelica Conference*, pages 541–550, 2009.
- [2] J. Biesiadecki, A. Jain, and M. James. Advanced simulation environment for autonomous spacecraft. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, (Tokyo, Japan), 1997.
- [3] P. Bodin, M. Nylund, and M. Battelino. SATSIM - A real-time multi-satellite simulator for test and validation in formation flying projects. *Acta Astronautica*, 74:29–39, 2012.
- [4] K. Brieß, W. Bärwald, E. Gill, H. Kayal, O. Montenbruck, S. Montenegro, W. Halle, W. Skrbek, H. Studemund, T. Terzibaschian, and H. Venus. Technology demonstration by the BIRD-mission. *Acta Astronautica*, 56:57–63, 2005.
- [5] J. Carrico, D. Conway, D. Ginn, C. Folta, and K. Richon. Operational use of swingby—an interactive trajectory design and maneuver planning tool - for mission to the moon and beyond. In *Proceedings of the 1995 AAS/AIAA Astrodynamics Specialist (Halifax, Canada)*, 1995.
- [6] E. Dam, M. Koch, and M. Lillholm. Quaternions, interpolation and animation. Technical report, Department of Computer Science, University of Copenhagen, 1998.
- [7] C. de Boor. *A Practical Guide to Splines (Revised Edition)*. Springer-Verlag, 2001.
- [8] S. Eckert, S. Ritzmann, S. Roemer, and W. Bärwald. The TET-1 satellite bus a high reliability bus for earth observation, scientific and technology verification missions in leo. In *The 4S Symposium (Portugal)*, 2010.
- [9] F. Eke. Dynamics of variable mass systems. Technical report, Dep. of Mechanical and Aeronautical Engineering (Univ. of California), 1998.
- [10] F. Eke and T. Mao. On the dynamics of variable mass systems. *International Journal of Mechanical Engineering Education*, 30:123–137, 2000.
- [11] P. Ferguson, T. Yang, M. Tillerson, and J. How. New formation flying testbed for analyzing distributed estimation and control architectures. In *AIAA Guidance, Navigation, and Control Conference and Exhibit (Monterey)*, 2002.
- [12] A. Heckmann, M. Otter, S. Dietz, and J. Lopez. The DLR flexible bodies library to model large motions of beams and of flexible bodies exported from finite element programs. *The Modelica Association*, 2006.
- [13] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag London, 1995.
- [14] H. Joos, J. Bals, G. Looye, K. Schnepper, and A. Varga. A multi-objective optimisation based software environment for control systems design. *Proc. of 2002 IEEE International Conference on Control Applications and International Symposium on Computer Aided Control Systems Design, CCA/CACSD*, 2002.
- [15] G. Kaplan, J. Bangert, J. Bartlet, W. Puatua, and A. Monet. User guide to NOVAS 3.0. *USNO Circular 180*, 2009.
- [16] D. Klumpar, H. Spence, B. Larsen, J. Blake, L. Springer, A. Crew, E. Mosleh, and K. Mashburn. FIREBIRD: A dual satellite mission to examine the spatial and energy coherence scales of radiation belt electron microbursts. In *American Geophysical Union, Fall Meeting*, 2009.
- [17] W. Larson and J. Wertz, editors. *Space Mission Analysis and Design, 3rd edition*. Microcosm, 1999.
- [18] F. Lemoine, S. Kenyon, J. Factor, and R. Trimmer et al. The development of the joint nasa gsfc and nima geopotential model egm96. Technical report, NASA Goddard Space Flight Center, 1998.
- [19] M. Lovera. Object-oriented modelling of spacecraft attitude and orbit dynamics. In *54th International Astronautical Congress, Bremen, Germany*, 2003.
- [20] M. Lovera. Control-oriented modelling and simulation of spacecraft attitude and orbit dynamics. *Journal of Mathematical and Computer Modelling of Dynamical Systems*, 12:73–88, 2006.

- [21] S. Mattsson and G. Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal of Scientific and Statistical Computing*, 14:677–692, 1993.
- [22] S. Maus, S. Macmillan, S. McLean, B. Hamilton, A. Thomson, M. Nair, and C. Rollins. The US/UK world magnetic model for 2010-2015. Technical report, NOAA, NESDIS/NGDC, 2010.
- [23] Modelica Association, editor. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.2*. 2010.
- [24] O. Montenbruck and E. Gill. *Satellite Orbits - Models, Methods, and Applications*. Springer Verlag, Heidelberg, 2000.
- [25] M. Otter, H. Elmqvist, and S. Mattsson. The new modelica multibody library. *Proceedings of the 3rd International Modelica Conference*, 2003.
- [26] C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal of Scientific and Statistical Computing*, 9:213–231, 1988.
- [27] A. Pfeiffer. Optimization library for interactive multi-criteria optimization tasks. In *9th International Modelica Conference*, 2012.
- [28] J. Picone, A. Hedin, and D. Dro. NRL-MSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *Journal of Geophysical Research*, 2001.
- [29] T. Pulecchi, F. Casella, and M. Lovera. A modelica library for space flight dynamics. In *Proceedings of the 5th International Modelica Conference*, 2006.
- [30] T. Pulecchi, F. Casella, and M. Lovera. Object-oriented modelling for spacecraft dynamics: Tools and applications. *Simulation Modelling Practice and Theory*, 18(1):63 – 86, 2010.
- [31] M. Reiner. *Modellierung und Steuerung eines strukturelastischen Roboters*. PhD thesis, Technische Universität München, 2011.
- [32] M. Reiner, M. Otter, and H. Ulbrich. Modeling and feed-forward control of structural elastic robots. *AIP Conf. Proceedings of the Int. Con. of Numerical and Analysis and Applied Mathematics*, pages 378–381, 2010.
- [33] F. Schiavo and M. Lovera. Modelling, simulation and control of spacecraft with flexible appendages. In *5th International Symposium on Mathematical Modelling, Wien, Austria*, 2006.
- [34] C. Schweiger and M. Otter. Modeling 3D mechanical effects of 1D powertrains. *Proceedings of the third International Modelica Conference*, 2003.
- [35] M. Thümmel, G. Looye, M. Kurze, M. Otter, and J. Bals. Nonlinear inverse models for control. *Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8*, 2005.
- [36] A. Turner. An open-source, extensible spacecraft simulation and modeling environment framework. Master’s thesis, Virginia Polytechnic Institute, 2003.
- [37] US Nautical Almanac Office. *The Astronomical Almanac for the Year 2011*. United Kingdom Hydrographic Office, 2010.
- [38] O. Wallrapp. Standardization of flexible body modeling in multibody system codes, part I: Definition of Standart Input Data. *Mechanical Structures & Machines*, 22(3):283–304, 1994.



# Modelica Stage Separation Dynamics Modeling for End-to-End Launch Vehicle Trajectory Simulations

Paul Acquatella B., and Matthias J. Reiner

German Aerospace Center (DLR), Institute of System Dynamics and Control  
82234 Wessling, Germany

paul.acquatella@dlr.de · matthias.reiner@dlr.de

## Abstract

Stage separation dynamics modeling is a critical capability of future launchers preparatory studies. The development of stage separation frameworks integrable in end-to-end launch vehicle trajectory simulations have been presented in the relevant literature but none profiting from the object-oriented and equation-based acausal modeling properties of MODELICA. The objective of this paper is therefore to present such an approach to this problematic. Based on the *Constraint Force Equation* (CFE) methodology, two case studies to evaluate the proposed approach are considered. Results demonstrate that the approach corresponds very well with the physics behind separation. In addition, we found easiness of implementation of the method within a single environment such as DYMOLA, demonstrating the benefits of an integrated approach.

## 1 Introduction

Stage separation dynamics modeling is a very challenging task and a critical capability that must be considered in the preparatory studies and development of next generation launchers [14, 16, 17]. The integration of such stage separation modeling into a single environment capable of end-to-end launch vehicle trajectory simulation is also a key technology to aim for.

The importance of such capability arises from the fact that after separation, the integrity of each stage must be kept in order to guarantee overall success of the space mission pursued. In this sense, the development of an integrated framework for analysis and simulation of stage separation is desired.

Early efforts on the subject of multi stage launch vehicle separation from the 60's and 70's are mainly

from NASA studies [1, 2, 4] and their *Program to Optimize Simulated Trajectories (POST)* as a generalized trajectory simulation and optimization software [5], developed in partnership with the (then) Martin Marietta Corporation. Renewed interest in the subject in the 2000's led NASA's development of a stage separation conceptual separation tool, *ConSep* [11, 12, 13, 14]; which is a MATLAB-based wrapper to the commercially available ADAMS solver, as its predecessor *SepSim*. However, being *SepSim* and *ConSep* dependent on the commercial software ADAMS, they have the disadvantage of not being easily integrable in a generic trajectory simulation software. This in turn eludes the capability of performing efficient end-to-end launch vehicle trajectory simulations. As a result, a generalized approach to stage separation problems of launch vehicles was developed [16]. The approach, coined as the *Constraint Force Equation* (CFE) methodology, was implemented into the *Program to Optimize Simulated Trajectories II (POST2)*, the *POST* follow-up. Separation studies applied to real platforms such as the Hyper-X or the Space Shuttle can be found in [18, 10]. The thesis [15] studies launcher separation analysis with OPENMODELICA but results in a tool (*OMSep*) which is only capable of input-output analyses at separation time, and not for generic launch vehicle trajectories.

As yet, an object-oriented and equation-based acausal modeling approach to stage separation dynamics integrable in end-to-end launch vehicle trajectory simulations is still missing. Such approach could potentially facilitate the integration of this and other capabilities within a single multi-physics environment such as DYMOLA.

The objective of this paper is therefore to present such an alternate approach to stage separation dynam-

ics based on the CFE methodology using MODELICA [7, 8]. We do this by means of the following sub-objectives: We study first the modeling challenges of multi-stage launcher separation dynamics; then we present an approach based on CFE implemented in MODELICA; following, we provide two case studies for which we apply the method; and finally we present some results and discussion, outlining benefits and disadvantages.

## 2 Modeling

For the simulation of launch vehicle stage separation dynamics, it is necessary being able to model two bodies connected together according to properly-selected constraints prior to their physical separation; and at the release command of such constraints, their subsequent and independent flight motion must continue. This section presents the separation dynamics and the separation mechanisms modeling aspects.

### 2.1 Separation dynamics

We refer to separation dynamics in this paper the study of the effects of forces and torques of a two-body system during their physical separation.

Such separation dynamics modeling clearly exhibits discontinuities similar to those described by other phenomena such as switching, limiting, friction, etc. Modeling must deal with these problems in special ways since this kind of behavior is sensitive to numerical solution errors, initial condition calculation/propagation, and integration in general.

MODELICA offers the possibility to implement a.o. several methods for such phenomena:

- *Stop and restart*: The complete system is simulated as a single body until separation time. Then the system is splitted into two bodies with independent states, and initial conditions are propagated accordingly. This solution however requires the split of two (or more) events.
- *Regularization*: This methodology consists on applying the constraint between the two bodies during their connected motion with a smooth but very stiff spring-damper system. This avoids the use of strict discrete or event behaviors. Such methodology is commonly used for simulation of friction, stiction, and other similar nonlinear behavior.

- *Hybrid*: This methodology consists on treating the simulation as a hybrid state machine where continuous and discontinuous behaviors are conditioned with data flows and proper transitions. This hybrid state machine framework is however complex to integrate in generic form for launch vehicle trajectory simulations.
- *Constraint Force Equation (CFE) Methodology*: The CFE methodology [16, 17, 18] consists on computing internal constraint forces and moments on two bodies during their connected motion and their application as external forces and torques to each of them separately. On separation command, these internal forces are set to zero, and then each body carries their own flight motion separately.

Of these methods, particular interest due to its applicability and easiness of implementation is given to the CFE methodology, which is selected as the primary method for the follow up of this study.

#### 2.1.1 Constraint Force Equation Methodology

The *Constraint Force Equation* (CFE) methodology [16, 17, 18] is a highly intuitive method consisting in the computation of joint loads, namely internal forces and torques, caused by joint constraints; along with their application as external forces and torques on each body independently, see Figure 1.

The joint loads which constrain one body’s motion relative to the other are dependent upon the external forces acting on each body as well as the type of joint. The net forces and torques on each body are therefore the sum of the usual external forces and torques *plus* the joint loads applied to each body as additional external forces and torques. In consequence, the CFE joint model simply augments the external loads of the system [17]. Quoting step by step [16, 17], the equa-

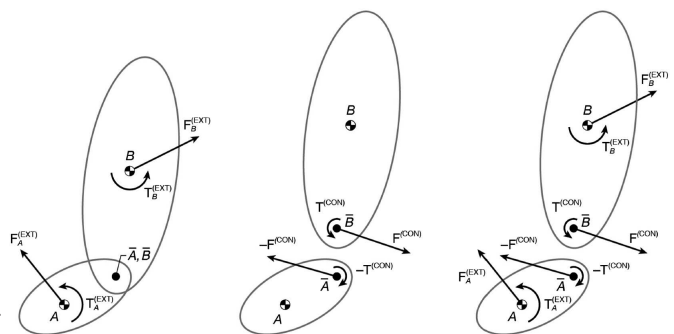


Figure 1: CFE diagram. Illustration credits: [16].

tions of constrained motion of two rigid bodies ( $A$  and  $B$ ) connected by a single joint (point  $\bar{A}$  in body  $A$  and point  $\bar{B}$  in body  $B$ ) are as follows:

$$\mathbf{F}_A^{(ext)} + \mathbf{F}_A^{(con)} = m_A \ddot{\mathbf{x}}_A, \quad (1)$$

$$\mathbf{T}_A^{(ext)} + \rho_A \mathbf{F}_A^{(con)} + \mathbf{T}_A^{(con)} = \mathbf{I}_A \dot{\boldsymbol{\omega}}_A + \boldsymbol{\omega}_A \times \mathbf{I}_A \boldsymbol{\omega}_A \quad (2)$$

where  $\rho_A$  is the position vector from the mass center of  $A$  to point  $\bar{A}$  of  $A$  at which the constraint force is applied. Similarly for  $B$ :

$$\mathbf{F}_B^{(ext)} + \mathbf{F}_B^{(con)} = m_B \ddot{\mathbf{x}}_B, \quad (3)$$

$$\mathbf{T}_B^{(ext)} + \rho_B \mathbf{F}_B^{(con)} + \mathbf{T}_B^{(con)} = \mathbf{I}_B \dot{\boldsymbol{\omega}}_B + \boldsymbol{\omega}_B \times \mathbf{I}_B \boldsymbol{\omega}_B. \quad (4)$$

There are so far 24 unknowns and 12 equations. Another set of six equations can be obtained from the law of action and reaction:

$$\mathbf{F}_A^{(con)} + \mathbf{F}_B^{(con)} = \mathbf{0} \quad (5)$$

$$\mathbf{T}_A^{(con)} + \mathbf{T}_B^{(con)} + (\mathbf{r}_B - \mathbf{r}_A) \times \mathbf{F}_B^{(con)} = \mathbf{0} \quad (6)$$

where  $\mathbf{r}_A = \mathbf{x}_A + \rho_A$  and  $\mathbf{r}_B = \mathbf{x}_B + \rho_B$ .

Six equations are missing. Worth noticing at this point, we only consider a *single joint* which constrain all six remaining degrees of freedom between the two bodies. This is because our focus is towards trajectory simulations and having multiple connections is not necessary unless when considering actuator sizing, sensitivity analyses, etc. In general, the CFE methodology allows to consider any type of joint which allows or not any specific relative motion between bodies; and redundancy of joints when necessary.

In this sense, for relative translation constraints and  $\mathbf{e}$  being unit-vectors of the corresponding ( $A$  or  $B$ ) body-frame, it is required that:

$$(\mathbf{r}_B - \mathbf{r}_A) \cdot \mathbf{e}_A = 0 \quad (7)$$

meaning that the distance between the two points of a particular direction remain fixed. And finally, for relative rotations constraints, it is required that:

$$\mathbf{e}_A \cdot \mathbf{e}_B = 0 \quad (8)$$

meaning that three properly selected two-unit-vector sets must remain perpendicular.

Eqs. (7)-(8) would have to be differentiated twice with respect to time so that the resulting relationships involve the unknown accelerations and angular accelerations, thus finally being able to couple them with the

equations of motion. In other words, the six missing equations are given by the following generalized constraint equations of the joint:

$$\ddot{g} = 0 \quad (9)$$

where  $g$  represents either of the nondifferentiated constraints in Eqs. (7) and (8). As it will be demonstrated in the next section, the manual differentiation of Eqs. (7)-(8) and their coupling with the equations of motion can be avoided altogether by the MODELICA implementation since this is done automatically. The last important aspect of the CFE methodology relevant to this work is the accuracy of the joint loads solution, which is sensitive to computational error and initial joint misalignment [17]. To handle such concern, the CFE algorithm could feature a.o. a stabilization technique known as Baumgarte stabilization [3, 6, 16]. This particular stabilization technique consists on replacing the ODE given by Eq. (9) which allows perturbations to grow linearly with time, by the following asymptotically stable ODE ( $\eta > 0$ ) involving terms of the once differentiated and nondifferentiated forms of  $g$ :

$$\ddot{g} + 2\eta \dot{g} + \eta^2 g = 0 \quad (10)$$

however at the expense of more computational effort. Many other stabilization techniques [6] could be implemented; these other methods, and a guidance for selecting  $\eta$  are however out of the scope of this paper.

## 2.2 Physical modeling of multi-stage separation mechanisms

Separation mechanism refers in this proposal to a mechanical model (or device) that makes separation possible in simulation (or reality). Physical modeling refers in this context on the capability to model separation behaviour by considering first principles (kinematics, dynamics, mechanics, physics, etc.); and being able to get realistic insight from such models for other purposes such as actuator sizing, sensitivity analyses, control, optimization, etc.

Based on our internal *DLR Space Systems Library*, separation mechanism physical models of different complexity levels can be studied. Simplified models for preliminary and conceptual studies; and more detailed ones for engineering validation aspects. These varying degrees of complexity would be helpful in order to perform separation mechanics analyses and

to assess the performance of the overall separation.

Configuration details of the separation mechanisms as well as their physical specifications must be provided to achieve more detailed and realistic models. Concerning the simple models, four variants have been studied:

- *Linear charge (release device)*: The linear charge model performs ideal or benchmark separation between two bodies. This mechanism “cuts” the two-body system on command. It simulates (ideal) explosive release devices, clamps, diaphragms, or point-release devices such as explosive bolts.
- *Bushing (separation impulse device)*: This model performs an impulsive reaction due to the release of a smooth but very stiff spring-damper system which keeps the two body system connected until separation command.
- *Kick-off spring (separation impulse device)*: Same as before, the impulsive reaction due to the release of a spring-damper system simulates the proper transmission of forces and moments of the two-body system during separation. This model is implemented with the *Constraint Force Equation* (CFE) methodology. This element is combined with a release device to simulate a realistic kick-off spring mechanism.
- *Generic (auxiliary devices)*: Other generic devices can be modeled in combination with the previous models, or with any other physical model from the library.

### 3 MODELICA implementation

In this section, the MODELICA implementation of separation mechanism models is presented. The challenges of this implementation strongly depends on the method selected as outlined in Section 2. Since the separation models in this work relies on a proper combination of the CFE methodology with physically-relevant elements, the implementation is not a straightforward application of existing MODELICA libraries; other aspects such as proper setup of initial conditions, state selection, modularity, and extendability are also challenging.

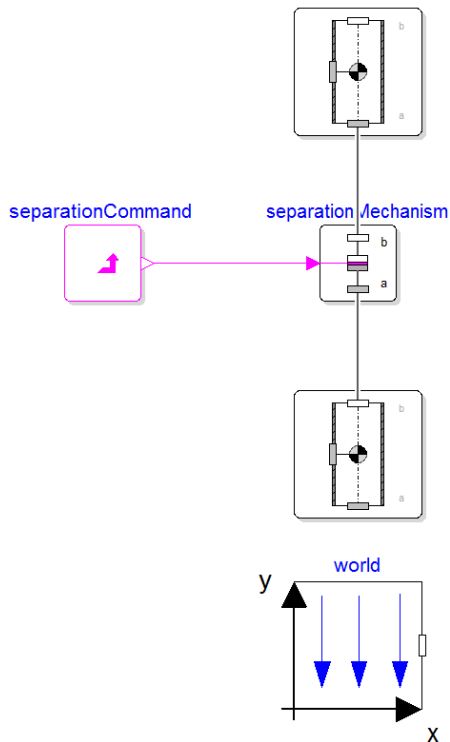


Figure 2: DYMOLA simulation layout consisting on a world model, two instances of rigid bodies, the separation mechanism model, and a boolean input for the separation command.

The baseline for the development of separation dynamics and separation mechanisms is the following partial mechanism model:

```

partial model PartialMechanism
  "Partial separation mechanism model"
  Interfaces.Frame_a frame_a
  "Joint frame a";
  Interfaces.Frame_b frame_b
  "Joint frame b";
  Interfaces.BooleanInput u;
end PartialMechanism;
    
```

As shown in the code, the partial mechanism interface model consists of two frames to connect a two-body system, and a boolean input for the ignition or separation command. Such interface allows the use of several separation models depending on the desired level of complexity by using replaceable instances. The approach here is bottom-up design, where the basis of separation dynamics simulation comes first from a single instance of a ‘release device’ mechanism.

In this work, a release mechanism model is implemented to simulate both a linear charge device com-



monly used in launcher stage separation, where the forces and moments at separation are zero; and as a base model for the next level of complexity. In other words, for the implementation of a separation impulsive device, an instance of a release device providing the capabilities of joint motion until separation is required on top of another physical model providing the corresponding impulsive forces or moments at the time of separation. Therefore, increasing the functionality to the separation model will consist on adding impulsive devices or simply improving the physics behind the device in question.

```

model SeparationMechanism
  "Separation mechanism model"
  Interfaces.Frame_a frame_a
    "Mechanism frame a";
  Interfaces.Frame_b frame_b
    "Mechanism frame b";
  Interfaces.BooleanInput u;
  replaceable Interfaces.PartialMechanism
end SeparationMechanism;

```

The implementation of the CFE procedure in MODELICA is as follows. The generalized constraint equations of the joint (9) have to be differentiated twice as explained before. Translational and rotational constraints at the joint are hence implemented as:

```

equation
  // generalized constraints
  g_con = frame_a.r_0 - frame_b.r_0;
  G_con = Frames.relativeRotation(frame_a.R
    , frame_b.R);

  // generalized velocity constraints
  g_con_dot = der(g_con);
  G_con_dot = Frames.angularVelocity2(G_con
    );

  // generalized acceleration constraints
  g_con_ddot = der(g_con_dot);
  G_con_ddot = der(G_con_dot);

  // CFE generalized joint constraints
  g_con_ddot = {0,0,0};
  G_con_ddot = {0,0,0};

```

In short, we present briefly two of the main models developed in this work:

- *Linear charge (separation release device)*: A release device is modeled by an instance of the *SeparationMechanism* model, called for instance **linearCharge**, which contains the partial interface outlined before, plus a switching mechanism between the CFE methodology and free body mo-

tion.

- *Kick-off spring (separation impulse device)*: An impulsive device is modeled by an instance of the *SeparationMechanism* model, called for instance **kickOffSpring**, which contains a **linearCharge** instance, plus a replaceable **separationMechanism** instance simulating the physics behind the impulsive device, such as a spring-damper system.

For a practical scenario to study, consider the trajectory phase of a generic launcher where the payload (Body *B* - the satellite to be placed in orbit) is to be separated from the remaining launcher upper stage (Body *A* - assuming a multi stage launcher). In this case, the problem consists of two bodies flying together under the effect of gravity in joint motion (the composite) up until separation is commanded. The separation command is usually given immediately after the shut down of the upper stage main engine. In this study however, we provide the separation command at any specified time. Figure 2 shows the DYMOLA simulation layout while Figure 3 shows a simulation of the physical setup of the case studies.

Initial conditions with respect to Earth-Centered-Inertial (ECI) frame of the composite are given to Body *A* as follows:

$$\mathbf{x}_A(t=0) = \begin{bmatrix} 1.1378 \times 10^7 \\ 0 \\ 0 \end{bmatrix} \text{ m,}$$

$$\mathbf{v}_A(t=0) = \begin{bmatrix} 0 \\ 5.9188 \times 10^3 \\ 0 \end{bmatrix} \text{ m/s}$$

and their translational and rotational dynamics are obtained from the rigid body model of the *Modelica Multibody Library* [9]. In the following section, we will study the separation dynamics implementation in MODELICA by means of two case studies: the first one considers the upper stage and payload (the composite) joint motion, while the second study considers the separation phase. For both cases, the forces due to gravity acceleration are obtained from the EGM96 model implemented in our internal *DLR Space Systems Library*. Both case studies are implemented in DYMOLA and the solution is computed using the DASSL solver with a tolerance of  $1e-7$ . A smaller tolerance of this solver would increase significantly the resulting chattering when Baumgarte stabilization is used.

**equation**

```

...
// CFE generalized joint constraints with Baumgarte stabilization
g_con_ddot + 2*eta*g_con_dot + eta*eta*g_con = {0,0,0};
G_con_ddot + 2*eta*G_con_dot + Frames.Orientation.equalityConstraint ( frame_a.R ,
frame_b.R ) = {0,0,0};
    
```

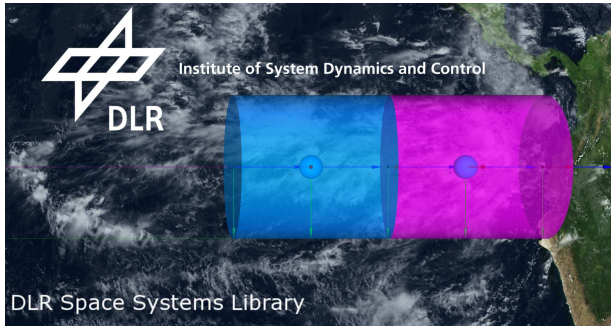


Figure 3: Simulation of the physical setup of the case studies.

Table 1: Mechanical properties of the two-body system.

Property	Body A	Body B	Units
Mass	6000	1000	Kg
$I_{11}$	23000	800	Kg·m <sup>2</sup>
$I_{22}$	23000	800	Kg·m <sup>2</sup>
$I_{33}$	18000	600	Kg·m <sup>2</sup>
$I_{21} = I_{31} = I_{32}$	0	0	Kg·m <sup>2</sup>

### 3.1 Case study I: upper stage and payload (composite) joint motion

The joint motion of the composite (Bodies A and B, the upper stage and the payload respectively) is simulated for a total time of 2000 s. During such motion, the MODELICA implementation of the CFE methodology is expected to derive automatically the joint constraint forces and torques such that the two-body system stays properly connected, with relative zero displacement. This case study therefore accounts for the validity of such implementation.

### 3.2 Case study II: upper stage payload separation dynamics

The upper stage payload separation is simulated in a practical scenario setup. It consists of a simulation of 20 s, half of which is in connected or joint motion, and then at  $t = 10$  s, the ignition command for separation is

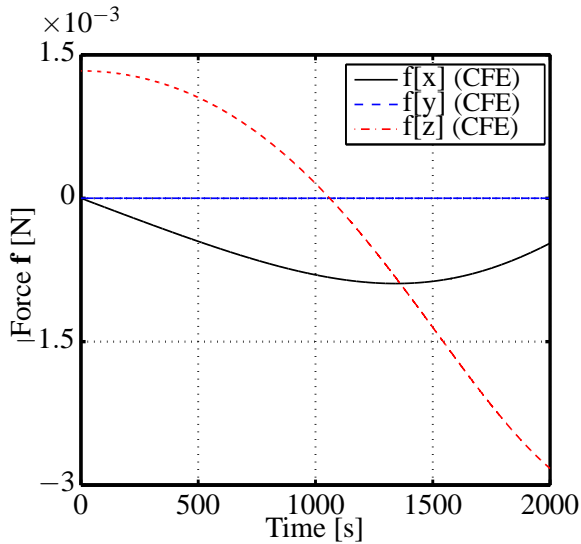
given. At this point, a kick-off spring separation mechanism model is in charge of the dynamical separation between the bodies. The subsequent independent motion of each body is then expected. This case study therefore accounts for the applicability of the physical models of separation mechanisms implemented.

## 4 Results and discussion

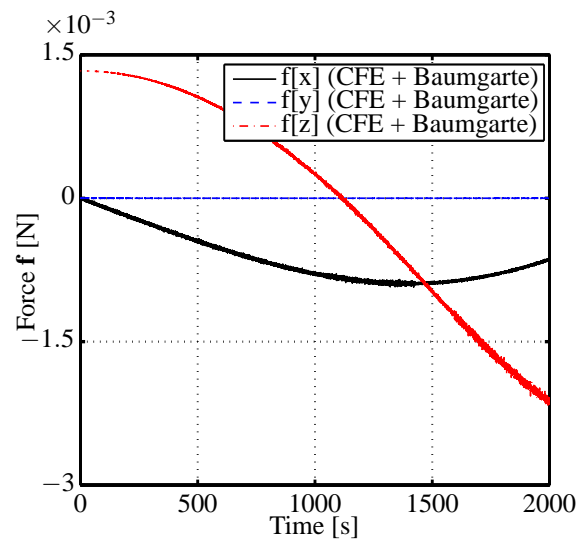
As outlined in the last section, **Case Study I** accounts for the study of internal forces and torques of the composite joint motion during a given portion of its trajectory by means of the *Constraint Force Methodology* implemented in MODELICA. During such joint motion, an important metric to assess the proposed method is the relative joint displacement between the two bodies when they are supposed to stay connected, as proposed and suggested by [17].

In this respect, Figure 4 presents the resulting constraint forces  $\mathbf{f}[i]$  and torques  $\boldsymbol{\tau}[i]$  at the joint during the connected motion, in all ECI directions  $i = x, y, z$ , respectively; while Figure 5 presents the resulting *relative* joint position  $\mathbf{rrel}[i]$  and the *relative* joint velocity  $\mathbf{vrel}[i]$ , in all ECI directions  $i = x, y, z$ , respectively.

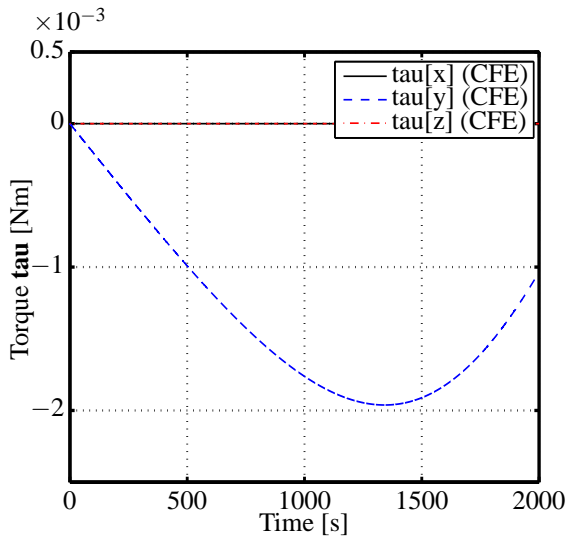
Results shows that the corresponding joint constraint forces and torques, obtained automatically by MODELICA in order to satisfy the CFE methodology constraints successfully keeps the bodies properly connected (hence, the composite) during their connected flight motion. Such result is evidenced looking at the relative joint position and relative joint velocity between the two bodies, which are supposed to be zero during the connected flight. A clear disadvantage for long simulation periods of joint composite motion is the necessity to keep the drift within physical boundaries, hence requiring a stabilization method. Stability and accuracy of the solution, especially for large simulation times, are improved with the addition of the *Baumgarte stabilization*. Nevertheless at the expense of chattering as shown in Figures 4-(b), 4-



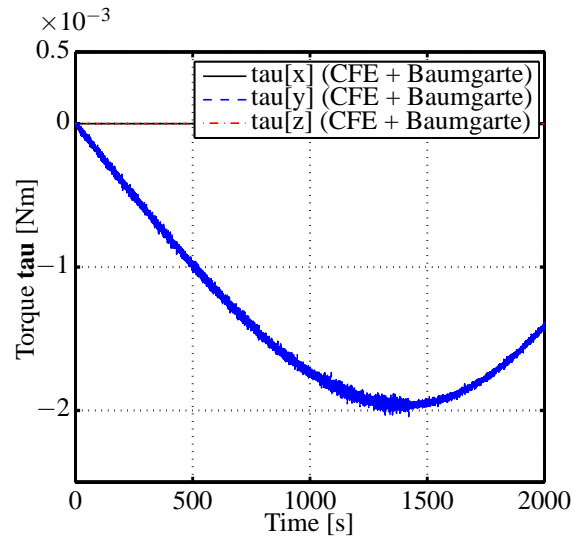
(a) Constraint forces at joint during connected motion with CFE methodology, in all ECI directions  $i = x, y, z$ .



(b) Constraint forces at joint during connected motion with CFE methodology plus Baumgarte stabilization with  $\eta = 2$ , in all ECI directions  $i = x, y, z$ .



(c) Constraint torques at joint during connected motion with CFE methodology, in all ECI directions  $i = x, y, z$ .



(d) Constraint torques at joint during connected motion with CFE methodology plus Baumgarte stabilization with  $\eta = 2$ , in all ECI directions  $i = x, y, z$ .

Figure 4: Case Study A results: constraint forces and torques at joint during connected motion.

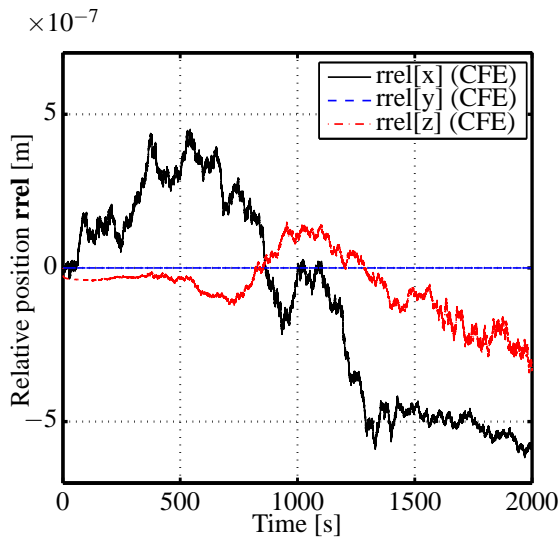
(d), 5-(b), 5-(d), meaning more computational time and effort.

**Case Study II**, as outlined in the last section, accounts for the study of absolute– and relative– position, velocity, and acceleration, respectively, between the two bodies from a multi-stage separation dynamics practical scenario. In here, the ‘release device’ simulated by a linear charge model has been augmented with an ‘impulsive device’ in parallel simulated by a kick-off spring model in order to simulate such a separation mechanism between the two bodies at their

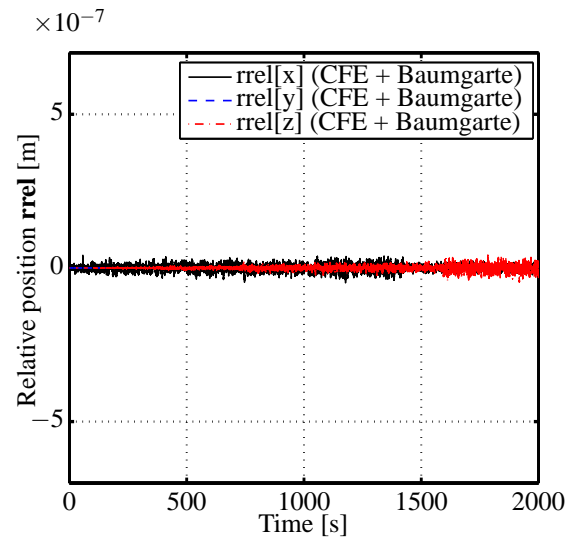
time of release from each other.

In this respect, Figure 6 presents the bodies’ *relative* position  $\mathbf{rrel}[i]$ , velocity  $\mathbf{vrel}[i]$ , and acceleration  $\mathbf{arel}[i]$  along the ECI orbital flight direction  $i = y$  (which is valid only for such a very small time frame) during the connected motion (first 10 seconds), and during their subsequent separation (last 10 seconds). Figure 6 also presents a zoom of the small time window just around the separation command.

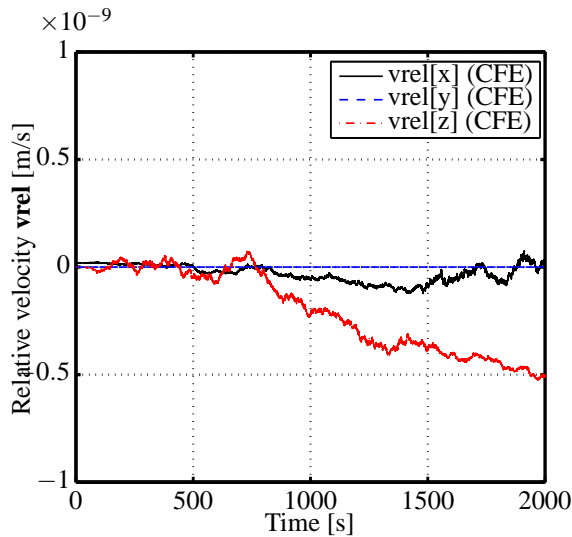
Results of this separation scenario shows the corre-



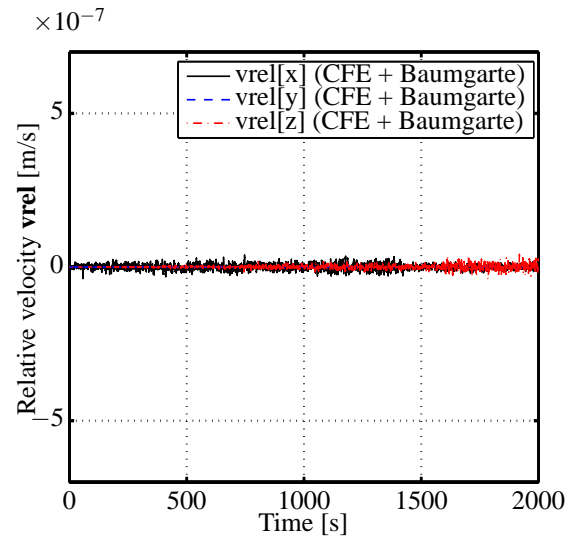
(a) Relative joint position during connected motion with CFE methodology, in all ECI directions  $i = x, y, z$ .



(b) Relative joint position during connected motion with CFE methodology plus Baumgarte stabilization with  $\eta = 2$ , in all ECI directions  $i = x, y, z$ .



(c) Relative joint velocity during connected motion with CFE methodology, in all ECI directions  $i = x, y, z$ .



(d) Relative joint velocity during connected motion with CFE methodology plus Baumgarte stabilization with  $\eta = 2$ , in all ECI directions  $i = x, y, z$ .

Figure 5: Case Study A results: relative joint position and velocity during connected motion, in all ECI directions  $i = x, y, z$ .

sponding relative states of the composite up until separation command and then their subsequent independent flight. Once again, the benefit and ease of use of the MODELICA implementation of the CFE methodology is evidenced during the connected flight of the composite, since constraint forces and torques are automatically computed and applied to the system. At separation, the relative states suggest an impulsive behaviour due to the kick-off spring separation mechanism model. This model releases a pre-compressed force stored in a replaceable spring-damper model, ev-

identing good correspondence with the physics behind separation. Such devices result in impulsive forces applied to the two-body system. This in turn causes a change in relative velocity and therefore, a successful physical separation of the system.

## 5 Conclusion

The objective of this paper was to present an object-oriented and equation-based acausal modeling approach to launch vehicle stage separation dynamics

with MODELICA. The aim is to develop an integrated approach for end-to-end launch vehicle trajectory simulation within a single environment.

Based on the *Constraint Force Equation* (CFE) methodology, two case studies to evaluate the proposed approach were considered. The scenario under study consisted of two bodies –representing a generic launcher stage and its payload– prior, during, and after their separation in orbital flight motion.

Results demonstrated that the approach, mainly thanks to the acausal and equation-based modeling features of the MODELICA language, corresponds very well with the physics behind separation while providing easiness of implementation within a single environment such as DYMOLA. The method computes and applies constraint loads automatically during joint motion and removes them accordingly at separation time, all in consistency with the CFE methodology.

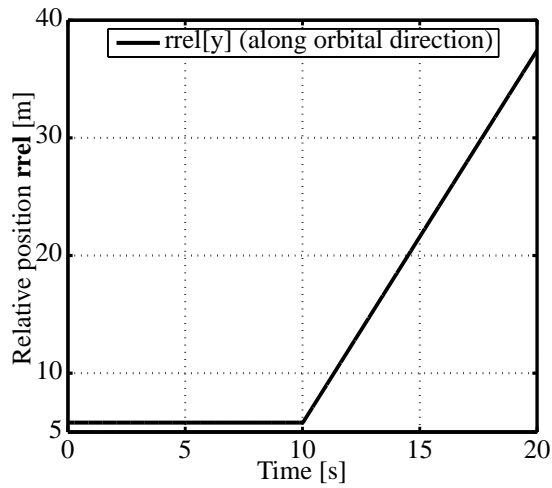
A disadvantage for long simulation periods of joint body motion is the necessity to keep the drift within physical boundaries, hence requiring a stabilization method. This in turn increases chattering and computational time and effort, thus resulting in a trade-off to consider for the task at hand. Validation studies are left to future work.

## Acknowledgements

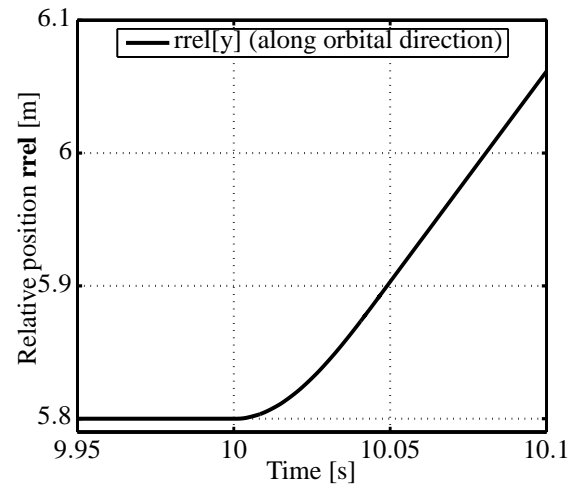
This work corresponds to DLR Institute of System Dynamics and Control activities of the ESA study **Upper Stage Attitude Control Design Framework** (USACDF), led by Astrium GmbH as part of Europe's Future Launchers Preparatory Program (FLPP).

## References

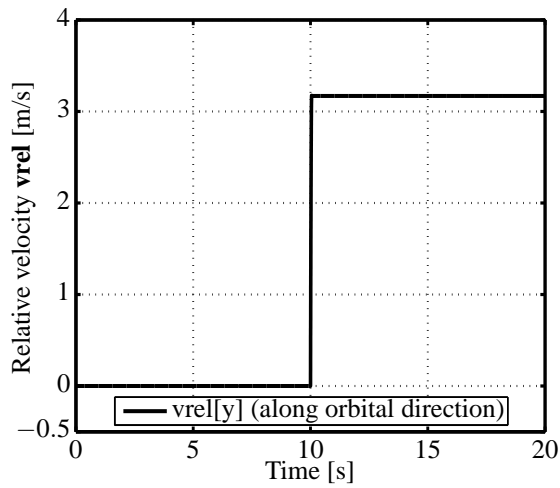
- [1] Decker, J. P. *Experimental Aerodynamics and Analysis of the Stage Separation of Reusable Launch Vehicles*. NASA-SP-148, January 1967.
- [2] Decker, J. P., and Gera, J. *An Exploratory Study of Parallel-Stage Separation of Reusable Launch Vehicles*. NASA TN D-4765, October 1968.
- [3] Baumgarte, J. *Stabilization of Constraints and Integrals of Motion in Dynamical Systems*. Computer Methods in Applied Mechanics and Engineering, Vol. 1, No. 1, 1972, pp. 1-16.
- [4] Decker, J.P., and Wilhite, A. W. *Technology and Methodology of Separating Two Similar Size Aerospace Vehicles Within the Atmosphere*. AIAA Paper 1975-29, Jan. 1975.
- [5] Bauer, G.L., Cornick, D.E., and Stevenson, R. *Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)*. NASA CR-2770, 1977.
- [6] Ascher, U.M., Chin, H., Petzold, L.R., and Reich, S. *Stabilization of constrained mechanical systems with DAEs and invariant manifolds*. Journal of Mechanics of Structures and Machines, Vol. 23, 1994, pp 135-157.
- [7] Elmquist, H., Mattson, S.E., and Otter, M. *Modelica - An International Effort to Design an Object-Oriented Modeling Language*. Summer Computer Simulation Conference, pp. 333-339, 1998, ISBN 1-56555-149-4.
- [8] Mattson, S.E., Elmquist, H., and Otter, M. *Physical system modeling with Modelica*. Control Engineering Practice, v. 6, pp. 501-510, 1998.
- [9] Otter, M., Elmquist, H., and Mattsson, S.E. *The New Modelica Multibody Library*. Proceedings of the 3rd International Modelica Conference, Linköping, 2003, pp 311-330.
- [10] Tartabini, P. V., Bose, D. M., McMinn, J. D., Martin, J. G., and Strovers, B. K. *Hyper-X Stage Separation Trajectory Validation Studies*. AIAA Paper 2003-5819, August 2003.
- [11] Murphy, K.J., Buning, P.G., Pamadi, B.N., Scallion, W.I., and Jones, K.M. *Status of Stage Separation Tool Development for Next Generation Launch Vehicle Technologies*. AIAA Paper 2004-2595.
- [12] Pamadi, B.N., Neiryneck, T. A., Covell, P.F., Hotchko, N.J., and Bose, D.M. *Simulation and Analyses of Staging Maneuvers of Next Generation Reusable Launch Vehicles*. AIAA Paper 2004-5185.
- [13] Pamadi, B.N., Hotchko, N.J., Jamshid Samareh, Covell, P.F., and Tartabini, P.V. *Simulation and Analyses of Multi-Body Separation in Launch Vehicle Staging Environment*. AIAA Paper 2006-8033.
- [14] Pamadi, B.N., Neiryneck, T. A., Hotchko, N.J., Scallion W.I., Murphy, K.J., and Covell, P.F. *Simulation and Analyses of Stage Separation of Two-Stage Reusable Launch Vehicles*. Journal of Spacecraft and Rockets, Vol. 44, No. 1, January-February 2007, pp 66-80.
- [15] Källdahl M. *Separation Analysis with OpenModelica*. Student thesis, Linköping University, Department of Electrical Engineering, Institutionen för systemteknik, Sweden, 2007.
- [16] Toniolo, M., Tartabini, P.V., and Pamadi, B.N. *Constraint Force Equation Methodology for Modeling Multi-Body Stage Separation Dynamics*. 46th AIAA Aerospace Sciences Meeting and Exhibit, 2008, 10.2514/6.2008-219.
- [17] Tartabini, P.V., Roithmayr, C.M., Toniolo, M.D., Karlgaard, C.D., and Pamadi, B. N. *Modeling Multibody Stage Separation Dynamics Using Constraint Force Equation Methodology*. Journal of Spacecraft and Rockets, Vol. 48, No. 4, July-August 2011, pp 573-583.
- [18] Pamadi, B.N., Tartabini, P.V., Toniolo, M.D., Roithmayr, C.M., Karlgaard, C.D., and Samareh, J.A. *Application of Constraint Force Equation Methodology for Launch Vehicle Stage Separation*. Journal of Spacecraft and Rockets, Vol. 50, No. 1, January-February 2013, pp 191-205.



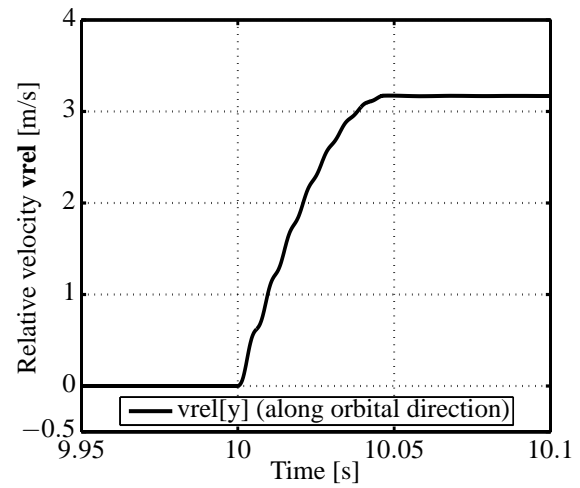
(a) Relative position between bodies A and B. The initial relative position (5.8 m) corresponds to the fixed distance between the bodies center of masses during joint motion.



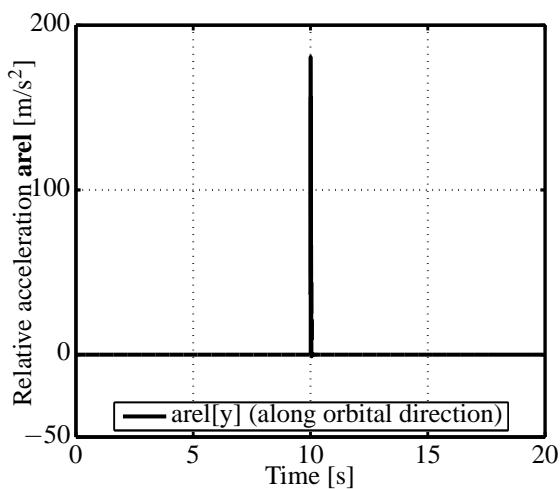
(b) Same as (a) with a close view around time of separation.



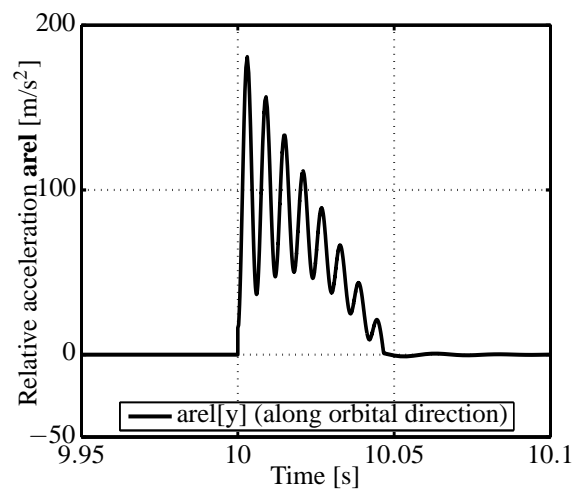
(c) Relative velocity between bodies A and B.



(d) Same as (c) with a close view around time of separation.



(e) Relative acceleration between bodies A and B.



(f) Same as (e) with a close view around time of separation.

Figure 6: Case Study B results: Relative position, velocity, and acceleration from a kick-off separation scenario along orbital flight direction  $i = 2$ . Ignition / separation command at  $t = 10$  s.

# A Modelica Library for Scalable Modelling of Aircraft Environmental Control Systems

Philip Jordan    Gerhard Schmitz

Hamburg University of Technology, Institute of Thermo-Fluid Dynamics  
Denickestr. 17, 21075 Hamburg

## Abstract

In the design process of complex technical systems such as environmental control and cooling systems for modern commercial aircraft, modelling and simulation is widely used. Simulation can provide viable insight during all phases of the system design cycle, but the questions to be answered by a simulation activity vary along the advancement of the design process. In this paper, the Modelica library developed in the Clean Sky project TEMPO is presented. The aim of the library is to support scalable system models which can be modified in detail and characteristic to be used during different phases of the design cycle without the need of rebuilding the system model or switching to another software tool. The library structure and the approach to integrate different detail levels is outlined and demonstrated at the example of a generic aircraft environmental control system architecture.

*Keywords: Environmental control system, scalable detail, object-oriented modelling, library*

## 1 Introduction

The competitive environment and the challenge to limit environmental impact while air traffic demand is increasing, aircraft manufacturers and suppliers are under continuous pressure to improve the performance of their products. Since it becomes increasingly difficult to improve the global system performance by independent optimization of subsystems, the need to instead aim for an optimized global system performance during development has been identified [1]. New and unconventional aircraft systems architectures such as the “More Electric Aircraft” (MEA) are the subject of past and ongoing European research programmes [2, 3].

The design process of complex technical systems such as aircraft environmental control and cooling systems can be broken down into characteristic design

phases. During all phases, modelling and simulation can be used to support the design process. However, each phase may pose different questions with regard to the physical phenomena of interest, the accuracy, or the simulation speed for example. Specialized tools can be used to meet the requirements for each simulation task. The library presented in this paper aims at using the object-oriented features of the Modelica language to integrate multiple layers of models. Each layer corresponds to a set of models designed for a specific simulation task and is subsequently referred to as “Detail Level”. The Detail Levels are integrated such that a single system model that has been assembled from available library component models can be “scaled”, i.e. reused, for a simulation activity of a different phase of the system design cycle.

The ability to create polymorphic models is a core feature of the Modelica language and it is used to create flexible and configurable models in many published Modelica libraries. An example are the variable dynamics settings and exchangeable heat transfer models in the `Dynami cPipe` Model of the Modelica Standard Library. Several libraries which pick up the concept and focus the design of the library more on scalable models have been presented.

In [4] structuring a Modelica library with scalable models for building simulation with the need for different models along the temporal evolution of a project in mind is discussed. A library for power plant simulation currently under development uses the idea of detail levels as group of models with similar complexity and incorporates the concept to organize the models within the library structure [5].

A somewhat different approach towards the concept of a “Detail Level” is taken in the library subject to this paper. Rather than a specific parametrization of a polymorphic model, a “Detail Level” represents the required characteristics of a model designed for a specific simulation task. This task is to be carried out at a certain point or in the system development process.

Switching a system model between two “Detail Levels” can therefore be seen as switching between different use cases where the information about the existing system structure is kept. The advantage of this is that the user does not have to rebuild the same system with another set of specialized models even though the actual model equations for each detail level may not have much in common.

The library discussed in this paper is developed within the scope of the TEMPO project (Thermal Exchange Modelling and Power Optimization). TEMPO is a research project in the Systems for Green Operations (SGO) technology demonstrator of the Clean Sky Joint Technology Initiative. The focus of the SGO demonstrator is to mature technologies which were developed during the MOET (More Open Electrical Technologies) project [2], and demonstrate architectural integration of thermal management technologies such as the electrically driven air system which was developed in MOET. In order to optimize and validate such architectures, extensive modelling studies are carried out. The presented library supplies the models for the environmental control system for the SGO demonstrator.

## 2 Models for Different Stages in the System Design Process

The process of designing and implementing a technical system is characterized by distinctive phases. In each phase design decisions have to be made and predefined milestones have to be reached before the development can proceed to the next phase [6]. The V-Diagram shown in figure 1 is a graphical representation of the system development process.

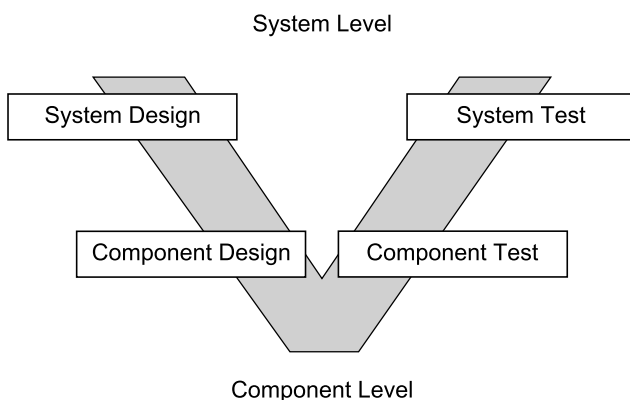


Figure 1: System design process displayed in the form of a V-Diagram

In this case, the process has been broken down into

four characteristic phases, System Design, Component Design, Component Test and System Test. Based on the tasks associated to the design phase, requirements for the models in the respective phase have to be formulated. The following section describes the characteristics of the Detail Levels of the TEMPO library. Table 1 gives a brief overview over characteristic differences of the main components for different Detail Level settings.

### 2.1 Phase 1: System Design

The system development process begins with the conceptual design of the system followed by preliminary system design. Based on customer need, system requirements have to be developed. From the overall system-level requirements, design requirements for the subsystem level are deducted. The phase covers feasibility assessments and leads to the decision on a system architecture. Simulation during this phase is used to support decisions on a system architecture and to deduct performance requirements for the components of the system.

The models designated for use during this phase are collected in the Detail Level 1 of the library. The Detail Level 1 models do not include dynamic effects. Performance characteristics are modelled by parameter setting. Turbomachinery components for example allow user defined efficiency settings, heat exchanger models are effectiveness based where the effectiveness can be a user defined parameter. The simple modelling assumptions aim for very fast and robust models for initial requirements estimation. The effects of a parameter variation on the system behaviour in this setting is limited by the constraints imposed on the remaining components.

### 2.2 Phases 2 and 3: Component Design and Component Testing

In the detailed design and development phase of the system development cycle, subcomponents and subsystems are brought into realization. As the design of the component progresses, more reliable performance predictions become available. These component performance predictions can be fed back into the system model, for example to check for compliance with overall system performance requirements. In parallel to the development of the actual components and their thermodynamic performance validation, the development of a control strategy is started. Models for control



Table 1: Overview of main component model characteristics for different Detail Level settings

Component	Detail Level 1	Detail Level 2	Detail Level 3	Detail Level 4
Turbine	Fixed operating point (parameter)	Map based, input filtering	Map based, input filtering	Simplified flow model, forward computation
Compressor	Fixed operating point (parameter)	Map based	Map based	Map based
Heat Exchangers	Static balance eqns, fixed effectiveness, optional local over/under determination	Static balance eqns, map based heat transfer, optional local over/under determination	Dynamic balance eqns, dynamic wall, geometric parametrization, discretization	Surrogate function for outlet temperatures calculation
Valves	Optional local under determination	Optional local under determination, input filtering, flow reversal supported	Input filtering	Input filtering, forward computation
Reservoir (tank) models	Pressure boundary, optional local over/under determination	Map based, optional local over/under determination	Map based, dynamic balances equations, global mass control	Map based, dynamic balances equations, global mass control

developments focus on the dynamic behaviour of the components.

When component prototypes have been produced or acquired, they are tested and additional performance data becomes available. In the form of performance maps, this information can be fed back into the system model, both for thermodynamic performance models and models for control development.

The Detail Level 2 Models of the library are designed to be used for thermodynamic performance simulations during these phases of the development process. The components are static models, performance characteristics are typically calculated from performance maps. Turbomachinery components use efficiency and flow characteristic maps, heat exchangers use effectiveness maps.

In the library, the models for control development activities during these phases are collected in Detail Level 3. These models use a more physical modelling approach. The heat exchanger models now include dynamic mass and energy balances on the fluid side and dynamic energy balances for the core material. Heat exchangers with connection to the incompressible liquid fluid domain also have the option to activate a dynamic momentum balance. The balance equations are discretized into finite volume elements. To somewhat limit computational effort and to improve robustness, the heat exchangers use lumped momentum balances and flow reversal is not supported. The parametrization of the heat exchangers is geometry based and can be modified to change surface geometries and flow patterns.

### 2.3 Phase 4: System Test

After the components and sub-systems have been designed and the prototypes have been manufactured and tested, they are integrated into the final system prototype. Simulation activities may include analysis of component behaviour with integration effects taken into account or hardware in the loop simulations to validate the developed control system.

Models for this kind of activity make up the Detail Level 4 in the library. The models used include dynamic behaviour, but the models are simplified in favour of faster simulation speed. The models are designed towards the use of fixed step solvers, while actual hardware-in-the-loop capabilities have not been implemented within the scope of TEMPO. A heat exchanger model of this Detail Level calculates outlet temperatures based on transfer functions. Fast states such as pressures in air components have to be removed to allow the usage of a sufficiently large step size with a fixed step solver.

## 3 Integration of Detail Levels

Two aspects regarding model structure for scalable models are discussed in this paper. This section treats how the detail level specific models are combined to achieve the model switching capability on the system level. Section 4 contains a description of the organisation of the models within the library structure.

The different available Detail Levels are collected on

the component level. Components in this context are units such as heat exchangers, valves or compressors. To create a system model, scalable components from the library are used to build up the desired architecture. As a sum of its components, the system model is also scalable. Scaling the system model is done by switching all components to a different Detail Level using a global parameter, while the system architecture information is kept.

In order to change the Detail Level setting of a system model, the `inner/outer` construct is used. The system model contains an `inner`-object of the class `SystemState`, in which the Detail Level setting is stored. In the library, the default instance name of the `SystemState` instance is “system”. All component models inherit from a `PartialComponent`-class which provides the interface to the `inner`-object “system” on the system level. In this class, the user can access the Detail Level setting from the graphical user interface.

### 3.1 Component Level Integration of Detail Levels

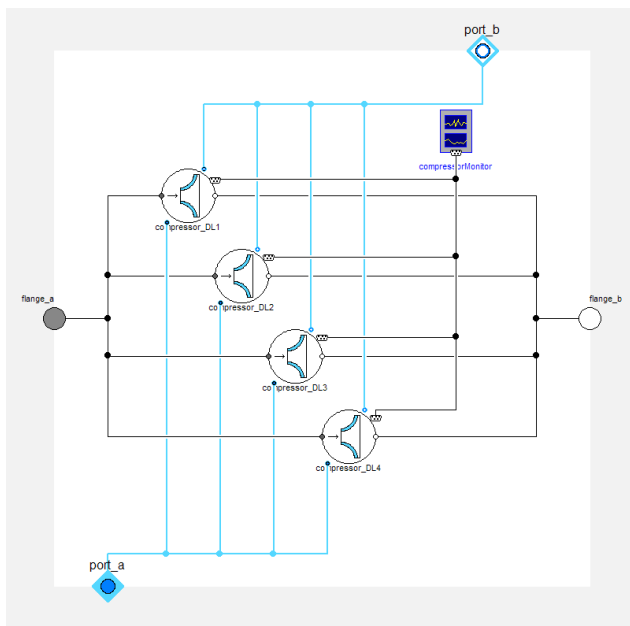


Figure 2: Container model structure seen as in the diagram layer view of a scalable component model

Polymorphism in Modelica is usually achieved by using the `redeclare` and `replaceable` language elements. Variations of encapsulated elements of the model can be exchanged in order to create the desired variant of the component. However, the Modelica language does not support parameter-based redeclaration. As a consequence, it is not possible to invoke a series of

redeclarations by one central parameter setting. Therefore `redeclare` and `replaceable` constructs can not be used to achieve the Detail Level switching behaviour needed for this application.

Instead the container method of class parametrization is applied to build scalable components from the Detail Level specific models. The container approach is a very basic parametrization method, where the scalable component model contains a set of conditionally declared variants of the component. Figure 2 shows the diagram layer of a scalable compressor model. Depending on the Detail Level parameter setting, only the desired model is instantiated. An advantage of this approach is the complete encapsulation of the Detail Level specific code. Model equations and class structure (internal class hierarchy levels for example) of a component may vary greatly between Detail Levels. The encapsulation allows task-oriented modelling independent of requirements for other Detail Levels, as long as the interfaces are compatible.

By default, the components are configured to follow the global Detail Level setting. This global Detail Level setting can be manually overridden individually for each component.

### 3.2 System Level Integration of Detail Levels

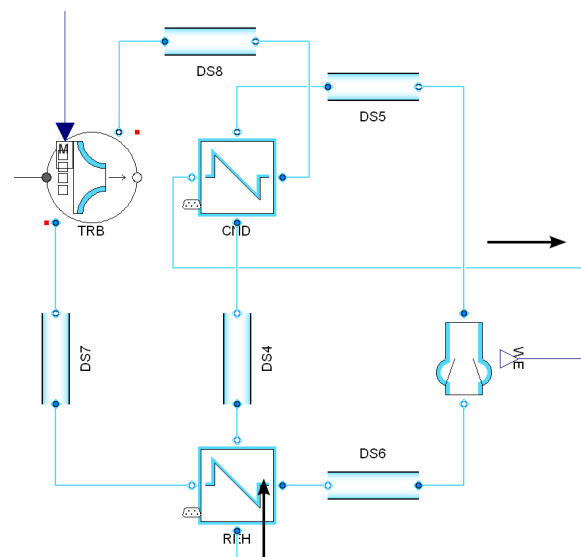


Figure 3: High pressure water extraction loop model with Detail Level dependent decoupling modules

A system model expressed in the equations of different Detail Levels can not always naturally be decomposed in the same manner. By building up a scalable system model from of scalable component models, at least one layer of decomposition is fixed and the library

has to address this issue. A common example of such a decomposition issue is a closed loop fluid system like the vapour cycle included in the application example in section 5. With dynamic mass and energy balances on the fluid side, the system is well-posed if it is made by connection of balanced component models according to the Modelica specification. A purely static variant of the same system on the other hand can not be created the same way. The connect-statement closing the loop introduces a redundant mass balance while the overall system lacks a “grounding”-equation.

A well-posed static model made from locally balanced components could be achieved by removing the connect statement and manually adding a pressure or total mass constraint. To be able to scale a system model without the need to modify the system topology, the option to add or remove equations has been shifted to the component models. The consequence is that individual components can now be configured to be locally over- or under-determined. With reference to the application example, this means that for the Detail Level 1 version of the vapour cycle, that the valve is configured to act as a loop breaker while the refrigerant reservoir is locally overdetermined and provides the necessary grounding. Local over- and under-determination is also used to adapt the causality of the overall system if required by the simulation task while the system level topology remains unmodified. In the presented vapour cycle system, the evaporator superheat is controlled by the expansion valve opening position. Be it that the control logic is not available and not to be simulated in the conceptual design phase (Detail Level 1). Instead, the evaporator superheat can be fixed. The additionally introduced equation in the evaporator model is compensated by configuring the valve coefficient as an unknown. This requires extra care by the user, since a globally balanced system is no longer guaranteed.

Another aspect which has to be considered for integrating the Detail Level specific models into a scalable system model are the constraints imposed by the system architecture information. When the Detail Level specific equations are incorporated into a scalable component model, the architecture of the system in which they will be used in is yet unknown. But the overall equation system which will have to be solved includes the constraints associated with the system architecture information. Unfavourable combinations of model equations and connections can lead to high index systems or non-linear algebraic loops. Such unfavourable equation system structure cannot always be avoided, but the modeller will generally try to adapt the system model and

the component configuration for improved robustness and performance. In thermo-fluid systems for example, complex networks of static components usually lead to large non-linear systems of equations. If the time scales of interest allow it, artificial dynamic states can be introduced to decouple the static components. In a scalable system model, it may occur that such measures are necessary for one Detail Level, but not for another. In a scalable system model, as it is presented here, multiple system models in fact “share” one set of system level constraints, i.e. the system architecture information. This means, that measures of which the necessity depends on Detail Level and system level constraints, need to be integrated into the Detail Level management. In the presented library, dynamic fluid states for decoupling large non-linear systems of equations are implemented in dedicated components.

The image in figure 3 shows the Modelica Diagram view of a model of the high pressure water extraction loop in air generation unit of an environmental control system (ECS). Such a system is installed on the Boeing 747-8 for example [7]. In a purely static configuration, the component equations together with the circular connections lead to a difficult to solve algebraic loop. The decoupling modules located between the components include the artificial state variables, but depending on the Detail Level setting, they are only activated when the Detail Level setting requires it.

## 4 Library Structure

The library structure is organized around component models such as valves or heat exchangers, which are ready to be connected in an executable system model after they have been parametrized. A general modelling philosophy is the aim for a flat inheritance structure for improved readability at the cost of possibly duplicate code elements. Where possible, the components contain no more than two inheritance levels. A common base class provides the component icons and common GUI elements like start value parameters.

The library covers three fluid domains: Air-, refrigerant- and incompressible liquid coolant domain. Thermofluidic components are by design permanently associated to a fluid domain. For example, there is a dedicated valve model for the air domain and a dedicated valve model for the incompressible liquid domain. The separation allows simpler—because more specialized—code for the components. The association of a component to a fluid domain is indicated by colour marking of the icon and connectors. The colour

of air domain components is blue, refrigerant domain components are identified by green coloured icons and connections and the incompressible liquid domain components are of orange colour.

### 4.1 Fluid Modelling

Within TEMPO, proprietary fluid property models provided by the SGO project partner are used. The data is accessed using external functions. The Modelica interface to the fluid data follows the Modelica.Media model structure for easy conversion to MSL fluid models. Each of the available fluid domain—air, refrigerant and incompressible liquid—have their own base package, fitted to the characteristics of the fluid type. The fluid connectors are identical to the Modelica.Fluid-connectors of the Standard Library. Currently, no distinction is made between total and static values for pressure and specific enthalpy.

Variants of the air type fluid model cover a range of dry and moist air models from simple and fast to more complex and accurate. Variants of the refrigerant and incompressible liquids cover several different fluids.

### 4.2 Component Organization

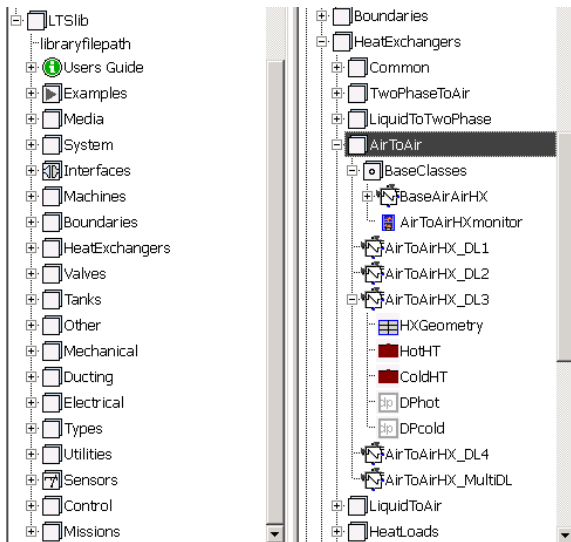


Figure 4: Package tree of the TEMPO library

Two main criteria are used to group the library elements in an efficient manner. The top level criterion is a generic functional classification of the type of component as shown on the left side of figure 4. Within each functional group, the components are classified according to the fluid domain they are associated with. On the right side of figure 4 it is illustrated for the HeatExchangers-package. All variants

of the Air-to-air heat exchanger model extend from a common base class. The scalable component model AirToAirHX\_MultiDL is the container model integrating all Detail Level specific models. The finite volume based dynamic heat exchanger model of Detail Level 3 is assembled from a set of subcomponents. These are located in the Common package.

## 5 Application to a System Model

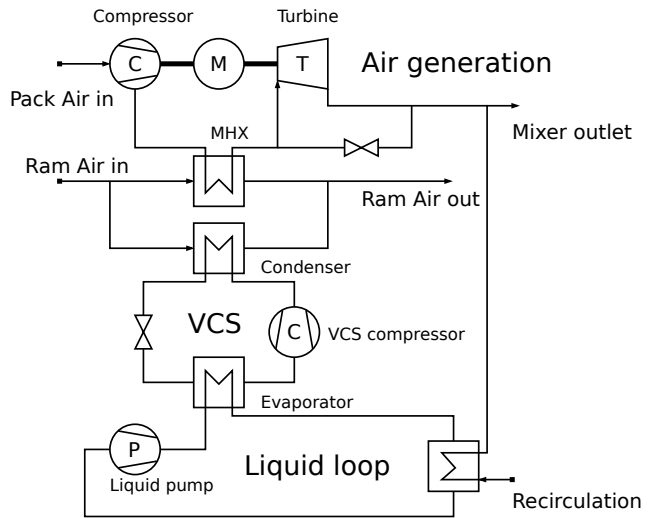


Figure 5: Reference architecture: Generic bootstrap cycle with motorized compressor, liquid cooling loop for cabin air recirculation cooling and vapour cycle

The application of the library is demonstrated on a generic ECS architecture. The architecture is modelled with the scalable components and simulated in different Detail Level settings. Fully real-time capable models which could be used in a real time environment are not yet supported by the library and are beyond the scope of TEMPO, therefore the simulated Detail Level settings cover the design phases 1–3 described in section 2.

The data used in the models such as performance maps and geometry information is proprietary, so that some the plotted results have been normalized.

### 5.1 Reference Architecture Description

The modelled ECS architecture is shown in figure 5. The system includes the air generation unit (pack) which provides pressurized air to the cabin. The presented system is based on a generic bootstrap cycle. Ambient air is pressurized in the compressor, excess heat is discharged in the main heat exchanger (MHX) using ram air as heat sink. The pack air is then cooled down

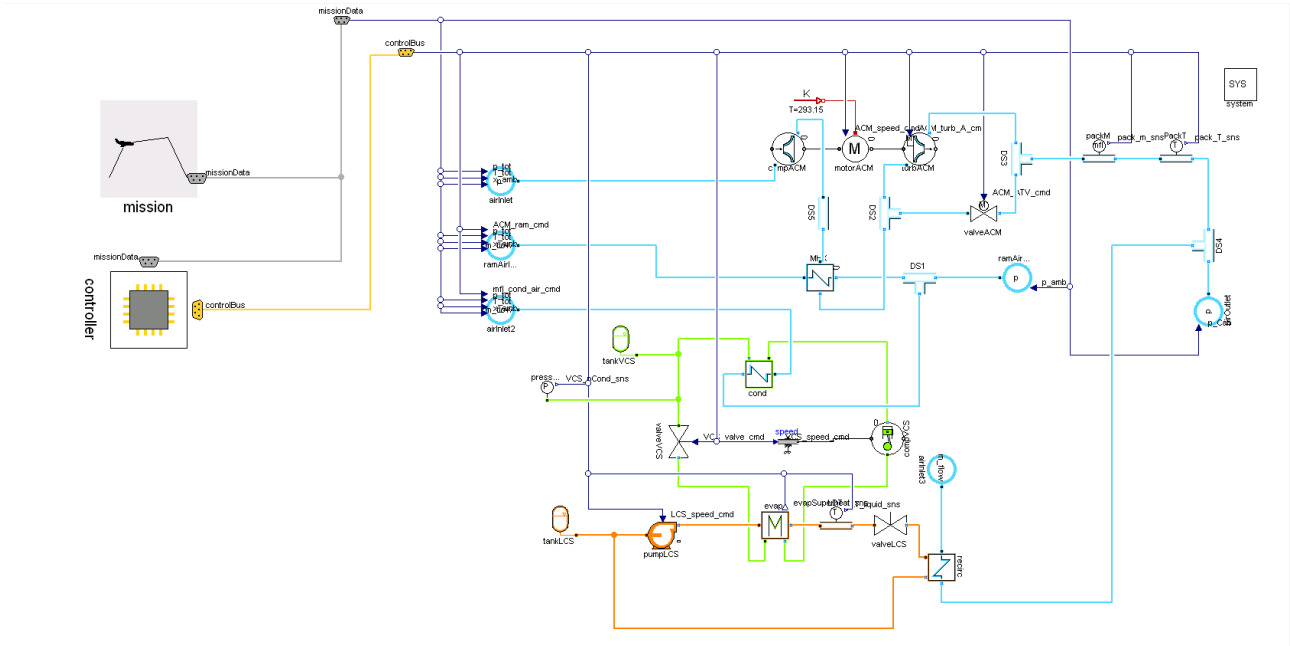


Figure 6: Diagram layer view of the ECS model

further in a turbine before it is released into the air distribution system. A liquid cooling loop is used to cool recirculation air from the cabin. A vapour cycle (VCS) is used to discharge the thermal load via ram air.

The diagram view of the reference architecture model is shown in figure 6. Boundary conditions and system control signals are provided by additional models. Mission (ambient conditions) and controller models are not integrated into the Detail Level management, i.e. the container model parametrization is not applied on them. The use of ambient data is independent from the Detail Level setting, therefore the data can be used in all Detail Levels. The controller model is assembled individually for a system as different controller models may be used on one ECS model.

In the model used for the simulations for the application example, the aircraft cabin is represented by an altitude dependent pressure boundary. The ram air mass flow control is simplified by directly imposing the mass flow rates and the cooling load from the recirculation air remains constant over time.

## 5.2 Detail Level 1 Use Case

All components of the system model are modelled statically. The model is intended for easy parametrization effects on the system and fast computation speed. This is reflected in simplified modelling assumptions and allowing to set properties as parameters that normally depend on the operating point. These proper-

ties are heat exchanger effectivenesses, superheating or subcooling temperature differences for vapour cycle heat exchangers, pressure ratios and efficiencies for turbomachinery or vapour cycle pressure levels for example.

For a simulation with the ECS model in Detail Level 1 configuration the ambient conditions for the model were set constant to represent a cruise flight at 38000 ft with Mach number  $Ma = 0.8$  at ISA conditions.

With the local over- and under-determination of the components, the degrees of freedom of the model can be changed. In the simulated example, the effectiveness as well as the superheat temperature difference of the VCS evaporator are fixed, so that the heat flow rate is determined twice. In return, the valve constant is set to be a free variable.

In figure 7 the evolution of the required condensation pressure and the power required is plotted for different settings of the condenser effectiveness. The pressure level has been normalized with a reference maximum pressure, the power demand has been normalized with a reference power.

## 5.3 Detail Level 2 Use Case

The Detail Level 2 configuration provides static models with more complex and accurate modelling assumptions. The components are mainly based on performance maps. Data available from detailed component design and prototype testing (Phases 2 and 3) can be

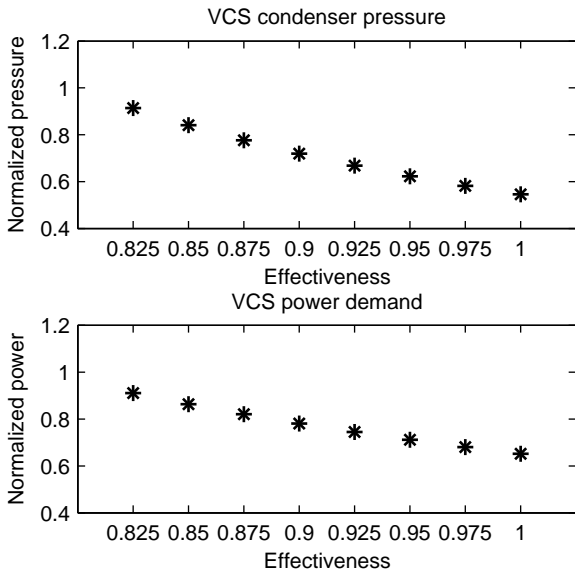


Figure 7: Evolution of required VCS high pressure level and VCS power demand for a given condenser effectiveness

used to verify the system performance. The additional variability of component performance properties allows to evaluate the effects of component interaction on the system level.

The ambient conditions for the Detail Level 2 simulation are unchanged and correspond to level flight at 38000 ft,  $Ma = 0.8$ . In this example, the vapour cycle superheat temperature is again set as a parameter. The ram air mass flow rate is reduced during the simulation. In figure 8 the output of the simulation is shown. From top to bottom, the curves show the evolution of the mass flow rates in the condenser, the normalized pressure levels of the VCS, evaporator liquid side outlet temperature deviation from target and the VCS power demand. The mass flow rates are normalized with respect to a reference mass flow rate, the pressure levels are shown in reference to a maximum allowed pressure. The result of the reduction of the ram air mass flow rate is an increase of the pressure levels. The change in evaporator pressure has an impact on the driving temperature difference and the evaporator heat flow rate decreases. As a result, the evaporator liquid outlet temperature increases. The result is presented in form of a deviation of a target temperature. In this case, the reduced ram air flow still provides enough cooling for the liquid cycle. However, the increased VCS power demand counteracts the reduced ram air flow.

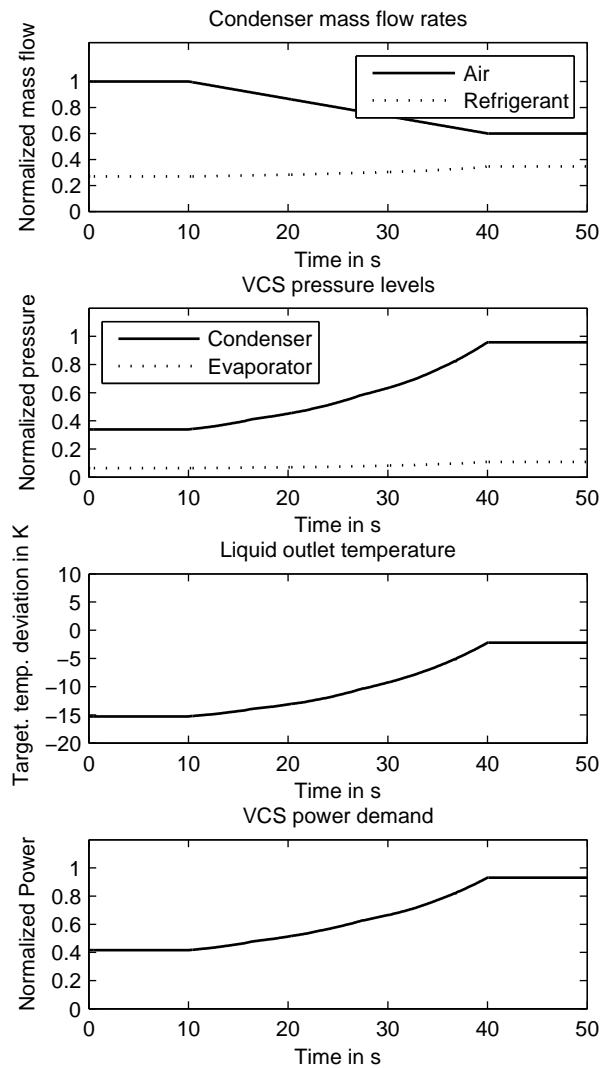


Figure 8: Results of simulation in Detail Level 2 setting

### 5.4 Detail Level 3 Use Case

The system model in Detail Level 3 setting switches the underlying equations to a combination of dynamic and static components. Valve, pump, turbine and compressor models are static and use performance maps similar to the Detail Level 2 setting. System dynamics are captured by the heat exchanger models. The liquid cycle components dynamic momentum balance option have been switched off.

The system model is now used to test a control strategy. In the Detail Level 3 example, pack outlet temperature, pack mass flow rate and evaporator liquid side outlet temperature are controlled to maintain target values. The controller model used together with the

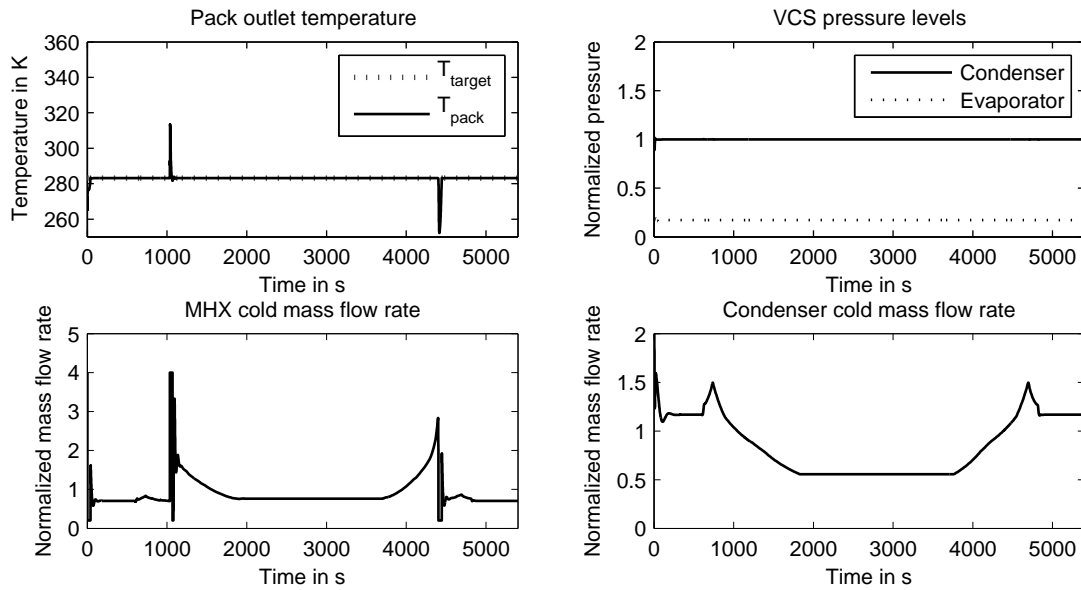


Figure 9: Dynamic mission simulation results

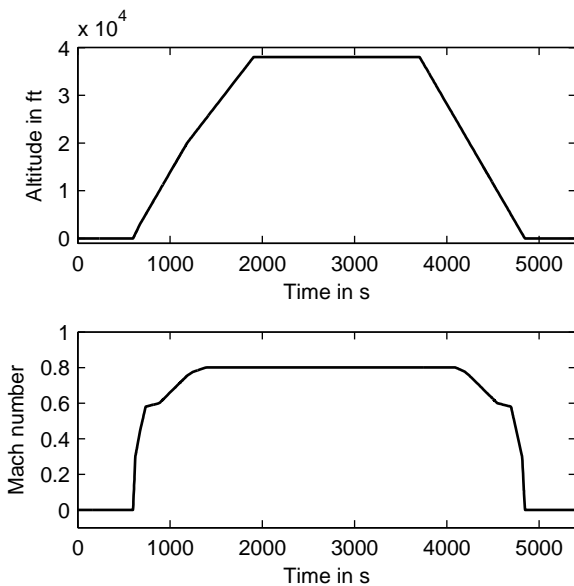


Figure 10: Mission altitude and Mach number profile for the dynamic Detail Level 3 use case

ECS model is assembled from has been implemented with elements from the Modelica Standard Library and the Modelica\_StateGraph2 library.

Figure 10 shows the mission profile which was used for the simulation. After an initial taxi-out period, a climb phase follows until a cruise altitude of 38 000 ft is reached. The descent begins after 30 min of cruise followed by a taxi-in phase on ground. The Mach profile is a function of the altitude, cruise speed is  $Ma = 0.8$ .

The top left graph in figure 9 displays the achieved pack outlet temperature, which matches the target output temperature except for two occasions. The bottom left graph shows the ram air mass flow rate in the main heat exchanger, which is modulated to control the pack outlet temperature. The mass flow rate shown is normalized by a reference mass flow rate. The controller tested in this simulation produces some overshoot and oscillation and should probably be revised and re-tested.

The control of the vapour cycle pressure levels however is successful in this simulation as it can be observed in the top right graph. The plotted data is normalized by the target condenser pressure. The bottom graph shows the condenser ram air flow which is used to maintain the condenser pressure. With increasing altitude, less ram air is needed due to the colder temperatures.

## 6 Conclusion and Future Work

In this paper, the Modelica library developed within the Clean Sky project TEMPO is presented. The library is designed for aircraft environmental control and cooling systems modelling. The focus of the library is the thermo-fluid domain, covering air, vapour cycle and liquid cooling systems. A key feature of the library is the integration of multiple levels of detail, where each Detail Level corresponds to a set of models specifically fitted to a certain simulation task. The detail levels are defined according to the simulation activities which are carried out along the process of developing

an environmental control system from the initial conceptual phase to the finished prototype. The library is organized around the Detail Level concept and allows to switch a system model from one Detail Level to another. The underlying model equations are thus exchanged to meet the requirements of another simulation task while preserving the information on the system architecture, freeing the developer of the need to entirely recreate the system model from scratch. As an application example, a scalable model of a basic aircraft environmental control system has been assembled and simulated for three different Detail Level settings. The development of the TEMPO library is ongoing work. A cabin model for commercial aircraft is currently under development. For the near future it is planned to make the library fully compatible to the recently extended Media-Library of the Modelica Standard library and to separate other protected intellectual property of the project partner from the core library. This version of the library will then be made available to the public.

## 7 Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) for the Clean Sky Joint Technology Initiative under grant agreement No. 270561.

## References

- [1] M. J. Provost, "The more electric aero-engine: A general overview from an engine manufacturer," in *Proceedings of the International Conference on Power Electronics, Machines and Drives*, Institution of Engineering and Technology (IET), 2002.
- [2] T. Jomier, "More open electrical technologies (MOET-0.02-AF-DEL-PublicReport-0001-09-R1.0)," technical report, MOET Consortium Partners, 2009. Available online at [http://www.eurtd.com/moet/PDF/MOET\\_Public\\_Technical\\_report.pdf](http://www.eurtd.com/moet/PDF/MOET_Public_Technical_report.pdf).
- [3] Clean Sky JTI, "Systems for green operations technology demonstrator overview." <http://www.cleansky.eu/content/project/system-green-operations>, accessed October 2013.
- [4] M. Bonvini and A. Leva, "Scalable-detail modular models for simulation studies on energy efficiency," in *Proceedings of the 8th International Modelica Conference, March 20th-22nd, Technical University, Dresden, Germany, 2011*.
- [5] J. Brunnemann, F. Gottelt, K. Wellner, A. Renz, A. Thüring, V. Roeder, C. Hasenbein, C. Schulze, G. Schmitz, and J. Eiden, "Status of ClaRaCCS: Modelling and simulation of coal-fired power plants with CO<sub>2</sub> capture," in *Proceedings of the 9th International Modelica Conference, September 3rd-5th, 2012, Munich, Germany, 2012*.
- [6] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*. Upper Saddle River, NJ: Prentice-Hall, 3 ed., 1998.
- [7] Boeing Corporation. [http://www.boeing.com/commercial/aeromagazine/articles/2012\\_q1/4/](http://www.boeing.com/commercial/aeromagazine/articles/2012_q1/4/), accessed in November 2013.



# Model-Based Energy Recuperation of Multi-Axis Machines

Tamás Juhász, Matthias Kennel, Marco Franke, Ulrich Schmucker

Fraunhofer Institute for Factory Operation and Automation IFF

Sandtorstr. 22, 39106 Magdeburg, Germany

{*tamas.juhasz, matthias.kennel, marco.franke, ulrich.schmucker*}@iff.fraunhofer.de

## Abstract

The peak power consumption of multi-axis production machinery (e.g. industrial robots) is determined by the forces during acceleration phases of continuous movements. The required high electric currents represent a cost factor in terms of the mains power supply. In this paper a new Modelica-based method is presented to save the mechanical braking energy of production machinery into a local flywheel-based energy recuperation system (ERS) for later utilization. An ERS has twofold advantages: on the one side it reduces the apparent power peaks from the mains power system. On the other side the overall energy consumption can also be reduced therewith. However, a mechanical ERS with a flywheel needs to be controlled in advance, as its internal inertia and the switched magnetic field introduce some dead time in the process. Therefore, the here presented approach uses an interdisciplinary Modelica model of the machinery to compute future power requirements prior execution of new movements. It is assumed that the machine program is known upfront in a textual form. The movement commands must be carried out with the virtual machine model first. The simulation computes the energy demand, according to which the stored amount of energy within the ERS (i.e. the angular velocity of its flywheel) must be controlled. The here computed reference angular velocity signal is put later also to the real controller, where the physical ERS is attached to as an extra motor. This paper presents the methodology along an example of a 3-axis robotic manipulator that is installed in the VDTC building of the Fraunhofer IFF, Magdeburg. This specific example has been used to validate the concept: 12% less power-consumption and 10% less power peaks were achieved during the operation.

*Keywords:* production machine, power efficiency, energy recuperation, flywheel, Modelica simulation, virtual NC

## 1 Introduction

Energy efficiency is nowadays a crucial competitive factor within the production industry. Additionally, in order to reduce production time, cycle times must always become shorter. The power requirement of production machinery is mostly dominated by the acceleration and braking of mechanical loads (e.g. heavy chassis parts in a car production line). As a common power supply for their induction motors, multi-axis machines are utilizing a single intermediate DC circuit. The motors of the machine are usually able to generate electric current upon external driving forces. This generated current can be recovered by the frequency inverters into the capacitors of the intermediate circuit.

The aim of this research was to develop a new methodology to integrate an electromechanical energy storage unit here, thus larger amounts of recovered energy can be utilized later in the production process again.

### 1.1 Energy Recuperation System - ERS

Modern high-speed flywheel-based energy recuperation systems (ERS) have high durability and can store a large amount of energy in a relatively compact space. A mechanical ERS itself is equipped with an AC induction machine that can also accelerate and decelerate the flywheel, thus convert between electrical and mechanical energy.

There are multiple industrial application fields where electromechanical ERS are being used with utmost efficiency. In case of production machinery they can be used as an uninterruptible power supply to overcome blackouts enabling the machinery to enter a safe state without causing any damage of the product. A most recent application is to compensate power fluctuations in electric networks of city trams [5]. This very application suggested the idea of this study: the integration of an ERS into a production machine.

## 1.2 Storing Energy in a Rotating Flywheel

The following formula can be used to compute the stored mechanical energy of a rotating flywheel:

$$E = \frac{1}{2} \theta \omega^2 \quad (1)$$

where  $\theta$  is the rotational inertia and  $\omega$  is the angular velocity of the flywheel rotor. The bearings of the flywheel are utmost optimized, thus the idling short-term friction losses are negligible. At ultra-high speeds (>25000 rpm, carbon-fiber flywheel) there is a need of a vacuum chamber to reduce drag losses. Nevertheless, the power consumption of this extra vacuum pump lessens the efficiency of the ERS.

There is a hurdle at using mechanical energy storage units in multi-axis machines due to large differences in mechanical and electrical time constants. The speed of an inert flywheel must be controlled in advance to be able to supply energy to and from the process effectively.

As production machines are usually program-driven, the next movement command can be known a priori. Therefore, the proper flywheel speed can be pre-computed by utilizing a mechatronic model of the whole system. The flywheel speed must be decreased a short time earlier than the axes begin to accelerate, and vice versa: it must be accelerated upon recuperating mechanical energy via the machine axes.

## 1.3 The ERMA Project

The here presented work has been done in the *ERMA* project (Energy Recuperation for Multi-Axis Machines) funded by the German Federal Ministry of Education and Research under the project number 02PK2192. The consortium agreed to demonstrate the methodology with a simple, 3-axis robotic manipulator that can be used to transport alloy wheels.

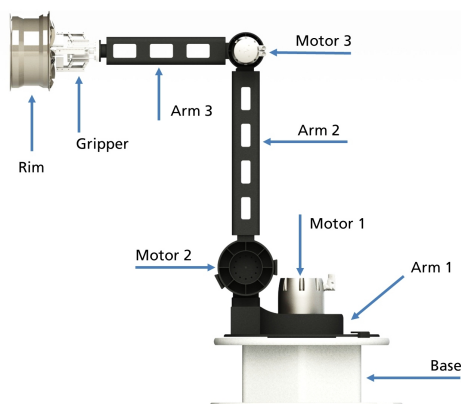


Figure 1: CAD model of the ERMA demonstrator

As production machinery in general is targeted by the here presented concept, a Siemens *Sinumerik*

*840D sl* NC controller was chosen to control the demonstrator. Numeric control is very common in the production industry and basically does not differ much from other interpreted robot programming languages. However, conventional NC controllers give no access to their internal axis interpolator states prior execution. Therefore, a look-ahead in the interpreted program is not directly possible.

Fortunately, Siemens offers a virtual NC kernel (VNCK), which is a software service that covers the whole functionality of a real Sinumerik NC interpreter offline, under Windows. An NC program (G-code) can be interpreted with the VNCK the same way as if it would run on the real controller, thus the reference trajectories can be extracted for each axis in advance. Therefore, it is known how a given movement command will be executed. The commissioning of VNCK is quite straightforward: once the configuration of the real Sinumerik NC has been set up with the real machine axes and parameters, the exported settings can be used later on demand to simply boot the VNCK. The identical configurations ensure a consistency between the simulated trajectories and the real executed ones.

The *ERMA* demonstrator system has been equipped with a prototype, steel flywheel-based ERS from the partner company *rosseta Technik GmbH* [5]. The ERS is shown on Figure 2 and has the following main properties:

- Mass: 52 kg
- Dimensions: 410 x 190 mm
- Induction machine: 20A peak current
- Rated power: 10 kW
- Rated energy: 22.8 kW<sub>s</sub>
- Max. speed 25000 rpm

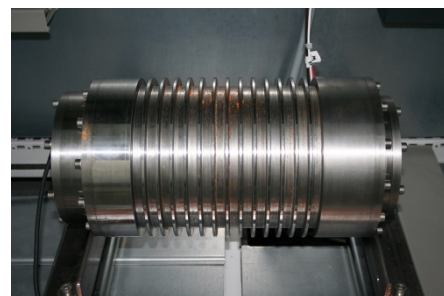


Figure 2: The flywheel-based ERS in the cabinet

The electrical cabinet including the four motor controllers (3 synchronous permanent-magnet drives of the manipulator + 1 asynchronous induction machine of the flywheel) have been developed by the partner company *aradex AG* [6].

## 2 Modeling in Modelica

In this chapter the modeling workflow is described along the example of the *ERMA* demonstrator system. It can be used to model a production machine conveniently in Modelica environment.

### 2.1 Mechanical Subsystem

In the recent years in the Fraunhofer IFF there has been a research focusing on automated translation of CAD information (geometry, physical properties and kinematic structure) into a parameterized Modelica multi-body description [1]. A plug-in was developed to generate MKS models directly from the Pro/Engineer CAD system. However, a lot of 3<sup>rd</sup> party Pro/Engineer assemblies could never be translated to a valid model without manual work. In a CAD environment the effort of defining the right constraints of a complex articulated mechanical system can be a troublesome work. In order to ease this task, designers usually circumvent the right definition of degrees-of-freedom between adjoined CAD assemblies and parts. Unfixed bodies or overdetermined constraints render the automated translation process erroneous. During the model conversion the constraints are normally mapped to Modelica joints automatically. Although the original constraints can be named, no meaningful joint names can be specified in Pro/Engineer. Therefore, the direct usability of a fully auto-generated model is questionable in model-in-the-loop scenarios.

Virtual commissioning of digital machine models with real controllers in the loop represents a new research focus in the Fraunhofer IFF. There must be a clear naming convention of joints to define the communication channels to and from simulated axes. Therefore, the solution of automated model translation had to be developed further. Using the *VINCENT* framework [2][7] the multi-body structure can be configured separately. The rigid bodies can be assembled together intuitively by using drag & drop from imported CAD assemblies and parts. The STEP interchange format is supported to let designers use various CAD systems that they have expertise in. Unfortunately there exists no widely-spread data exchange format yet, in which many CAD system developers would agree to include the constraints between the objects being designed. Therefore, the joints must be placed and connected with body elements in *VINCENT* manually, as well. However it seems to be a tedious work, after some practice a complete machine configuration can be defined with 40 bodies and 30 joints within a few hours. Once the structure has been configured, it can easily be reused

upon changing geometries or some other parameters. The Modelica model is directly generated out of the information in *VINCENT* using the same workflow as in [1]. Figure 3 shows the generated model according to the simple *ERMA* 3-axis robot. As long as the internal details are also modeled, the physical properties (mass, inertia, center of gravity) of rigid body elements with known uniform density can be simply derived from the fine-triangulated geometry models [3]. In case of simplified 3<sup>rd</sup> party CAD parts with no modeled interior the mass properties from datasheets of the manufacturer must be used instead.

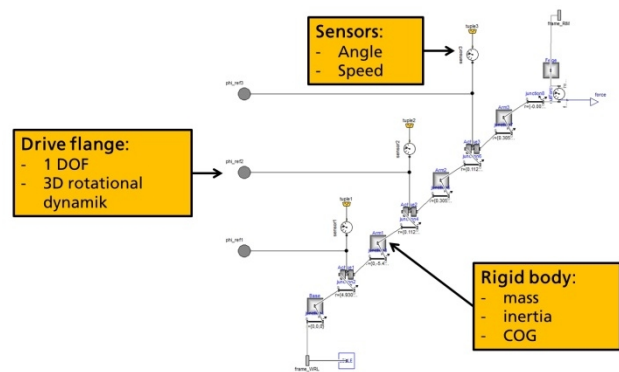


Figure 3: The generated multi-body model of the *ERMA* demonstrator

### 2.2 Usage of the Modelica Model during Sizing of the Drives

During its planned pick-and-place job of transporting alloy wheels the *ERMA* demonstrator must reach various points in space as fast as possible. At the early phase of the project the aforementioned multi-body model was used to compute acceleration limits. For this reason a new Modelica block was developed supporting point-to-point path interpolation of arbitrary goal positions within the workspace of a robot. Similar to the *KinematicPTP2* model in the standard Modelica library, this path interpolator allows the specification of kinematic constraints (maximum velocity and acceleration) of each output. Besides the start and end positions any number of intermediate locations can be given, too. As a result, time-optimal trajectories between adjacent goal positions are being computed.

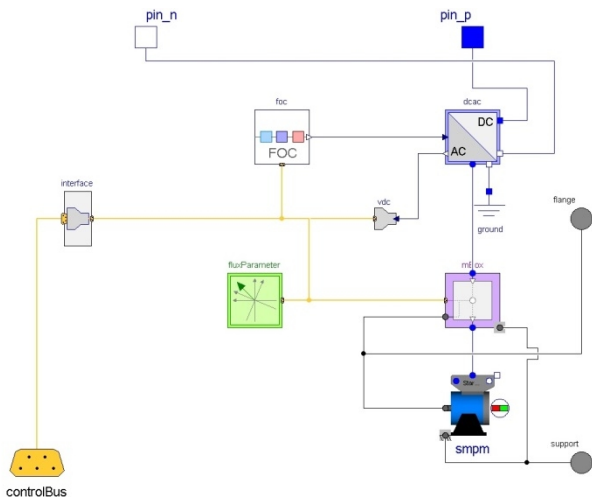
By using ideal position sources the simulator could compute the required torques during the sizing of the demonstrator. The simulated torque curves were taken to select the proper drive characteristics to be used. After multiple simulations involving iterative model updates using manufacturer data, a proper drive selection could be found. Table 1 summarizes the chosen drives including the gear ratios:

**Table 1:** Selection of Wittenstein-alpha permanent-magnet synchronous drives of the ERMA demonstrator

	Axis1	Axis2	Axis3
	TPM+	TPM+	TPM+
	power 110	power 110	high torque 025
Gear ratio (stages):	1:7 (1x)	1:35 (2x)	1:55 (2x)
Stall torque [Nm]:	(vertical)	878	214
Max. torque [Nm]:	600	1600	530

**2.3 Mechatronic Model of the Intermediate DC Circuit and the Electric Drive Subsystem**

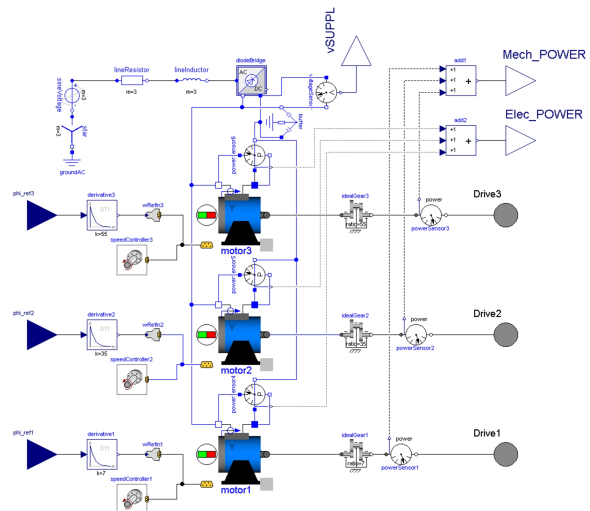
The prediction of future power requirements needs more than analyzing a pure mechanical model. Only an interdisciplinary model can compute the correct energy balance including the losses in other domains. The *Smart Electric Drives* Modelica Library has been developed at the *Austrian Institute of Technology* [8]. The SED library allows convenient modeling and simulation of an entire electric drive subsystem. It provides components of various electrical devices including motor models with power electronics and also energy storage modules. As both thermal and transient magnetic effects are taken into consideration, these models can fulfill the purpose of computing realistic power requirements. Figure 4 shows the diagram of a synchronous permanent magnet (SMPM) machine including field-oriented control and voltage and current limitations. A very similar control strategy is used in the real motor modules of the *ERMA* demonstrator.



**Figure 4:** Field-oriented controlled permanent magnet synchronous induction machine

This torque-controlled SMPM machine model from the SED library is to be connected to the intermediate DC supply circuit. It utilizes an internal DC/AC frequency converter and a three-phase synchronous induction machine model with permanent magnet excitation. The input torque is to be computed externally; a position controller can retrieve the required electrical state variables over a bus connector. The mechanical flange is connected to the respective revolutive joint in the mechanical model.

Based upon the specifications from the manufacturer the drive models were parameterized with real data. Measurements were carried out at the *aradex AG* to validate some model parameters such as winding resistances and inductances. The electric parameters of the AC/DC rectifier module were taken into account in a conventional three-phase diode bridge rectifier model from the SED library. It is extended with buffering capacitors to model the intermediate DC circuit that is built into the real electric cabinet. Figure 5 shows the diagram of the drive subsystem in Modelica:



**Figure 5:** Mechatronic model of the electric drive subsystem including the DC intermediate supply circuit

The drive subsystem model of the demonstrator includes a 400V three-phase AC power source, the DC circuit model and three synchronous permanent magnet motor blocks with an ideal gear and a position controller per each instance. The three-phase motor model uses space phasor formulation with d and q axes [4] and it considers thermal losses in the rotor bushing (mechanical friction) and electrical losses (eddy current, ohmic winding resistance), as well. Permanent magnet excitation is modeled by a constant equivalent excitation current feeding the d-axis:

$$I_e = \frac{1}{\sqrt{2}\pi} \frac{U_{OC}}{L_m a f_{nom}} \tag{2}$$

where  $U_{OC}$  is the RMS open circuit voltage at nominal speed,  $L_{md}$  is the main field inductance in d-axis and  $f_{nom}$  is the nominal frequency. These parameters could be found on the datasheet of the real motors. The here implemented flux-weakening control strategy weakens the magnetic rotor field so that the stator voltage is restrained to given voltage limitations even though the shaft speed rises.

The derivatives of the input axis position signals are fed to the proportional integral speed controllers. The internal gear ratio is taken here into consideration as a gain factor. The reference torque of each motor is computed according to the aforementioned field-oriented control. Feedback signals are available over the bus connector (see Figure 4).

The included standard Modelica power sensors allow measuring the momentary mechanical and electrical power consumption. The sums of those are the *POWER* outputs of this subsystem model. This way, after mathematical integration of the electric power signal, the energy consumption of arbitrary axis movements can be determined.

## 2.4 The Concept of the “Energy” Axis

In this work the energy demand of a production machine needs to be pre-computed with the presented mechatronic model according to original machine control programs. The energy demand within a small amount of time frame determines the next reference angular velocity of the flywheel in the ERS. For this reason an *ENERGY* spindle axis was defined in the Sinumerik controller of the *ERMA* demonstrator. This fourth axis represents the reference flywheel speed. Due to its inertia, the flywheel must accelerate earlier, in order to be able to recuperate the generated electric currents during a braking phase. However, the axes within the same channel of an NC controller are always controlled together as a bundle. There is no direct “predictive” way to let one of them move earlier than the others.

There is a solution to this problem by means of NC curve tables. A curve table is a way to look up and interpolate values of a *follower* machine axis based on the position of a *leader* axis. For this purpose there can be arbitrary pairs of leader-follower axis positions stored in the rows of a curve table. The only rule is that the leader positions must be increasing or decreasing monotonously. Upon moving the leader the NC controller interpolates a smooth movement of the follower. In the *ERMA* case the aforementioned *ENERGY* axis takes the role of the follower in each curve table definition. Upon each NC movement command there is at least one machine axis

involved. As the axis interpolation during movements is always monotonous, a leader machine axis can always be chosen respectively.

## 2.5 Coupling with the VNCK Service

As briefly introduced in chapter 1.2, there is a Windows service that emulates the core functionality of a real Sinumerik NC controller. The Virtual NC Kernel (VNCK) service registers a COM (Component-Object-Model) interface for easy integration into other Windows-based applications. Using this interface a Microsoft C# .NET-based GUI application called *IFF VNCK Manager* was developed. It can attach the Modelica simulation to the VNCK service and perform the transformation of an original NC program into a new form that includes the usage of curve tables to control the flywheel-based ERS.

Figure 6 shows the top-level view of the *ERMA* demonstrator model with VNCK coupling:

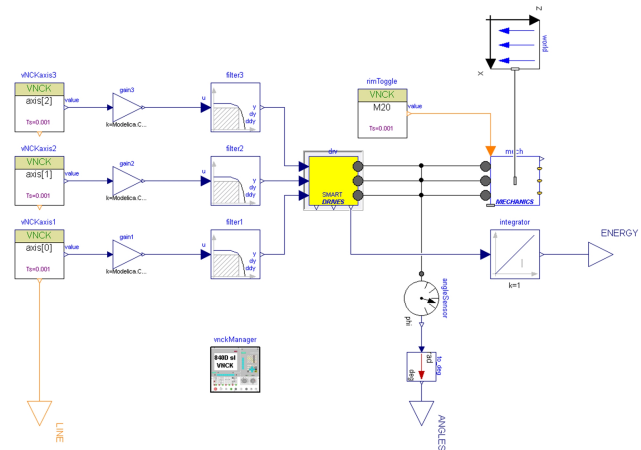


Figure 6: Top-level model of the *ERMA* demonstrator with VNCK binding

On the part of Modelica, a new *VNCK Library* has been created in the Fraunhofer IFF. It includes external C functions for binding, communicating with and finally stopping the VNCK service over a .NET/Managed C++ wrapper within the *ERMA* .NET solution. Modelica blocks were also developed for retrieving interpolated axis values and machine states at discrete time events. In the scenario of the demonstrator the latter way is used to detect commands for gripper control influencing the mass being transported. As the real gripper can programmatically get hold of or release a car wheel during the process, the load can vary over time. However, the virtual wheel body must always remain attached to the simulated robot’s gripper, because the model structure cannot be changed in Modelica during the simulation. Therefore a mass override input variable has been added to the standard Modelica rigid body

model. After a gripper release command is decoded by the VNCK, the mass of the wheel is set to zero immediately.

There is a single *VNCKManager* instance placed in this topmost Modelica model: the VNCK service is started during its initialization (in the initial equation section). The VNCK axis reader blocks are referencing this singleton instance over inner / outer Modelica constructs. During the processing of the original (textual) NC program, the VNCK is configured to report the source line positions, as well. This *LINE* integer output of the model is used to split the computed signals into multiple curve table definitions, as described later. According to an example NC program Figure 7 illustrates the splitting positions of the output signals (yielding six sections for six *G1* commands):

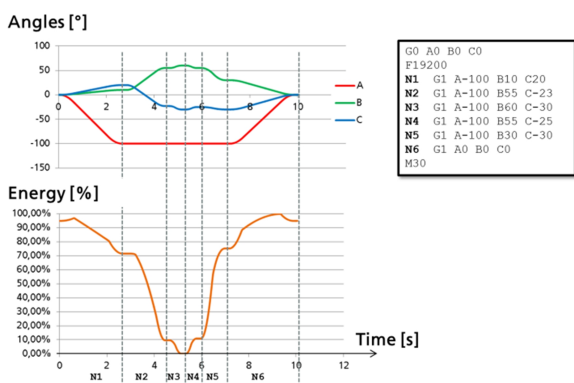


Figure 7: Splitting simulation signals according to NC statement blocks

The beginning of a new line with a movement command (in the example of Figure 7 those lines are labeled with *N1...N6*) is decoded by the VNCK and reported to the VNCK Manager.

The smallest step size of the VNCK service for axis-interpolation is 12 milliseconds. As only relatively coarse axis angles are being output therewith, filters must be applied in Modelica to smooth and reconstruct the continuous position reference signals. Each *VNCKaxis* block has a third-order Butterworth filter (10Hz cut-off frequency), and an additional first-order low-pass filter is applied in sequence, too. According to the smooth input signals the drive subsystem (Figure 5) can now compute the total power requirement at each simulation step (1ms). The smooth machine angles together with the integrated energy signal and the current NC source *LINE* signal are being output to the common simulation result file at equidistant intervals of 1ms.

The model of the machine being simulated can be arbitrary complex. There is no requirement of real-

time capability, because buffering mechanisms are used with semaphores within the .NET VNCK Manager application to synchronize the VNCK service and the Modelica simulation. The complete mechatronic model of the three-axis *ERMA* demonstrator is approximately 2 times slower than real-time. If the simulation would be faster and could overtake the VNCK process, it is forced to wait for the VNCK to yield the goal machine angles. For this reason the external C function call in each *VNCKaxis* can block the calling simulator thread until the buffer receives the next VNCK output (@ 83Hz / every 12msec) greater than or equal to the time step.

However, in case of *ERMA* the VNCK can compute three interpolated angles much faster than real-time. Therefore, the internal buffer of target angles is filled beforehand, thus buffer synchronization occurs only upon the beginning of the simulation.

The *VNCKManager* block shuts down the VNCK service and the simulation after receiving a *terminate* notification at the end of the interpreted NC program (*M30* command). After this event, the .NET application can start to process the simulation results.

## 2.6 Flywheel Control Strategy

Within the *ERMA* project a new strategy had to be developed to transform the computed energy levels to appropriate flywheel speeds, thus allow the ERS to reduce the power consumption of a production machine. There are multiple permanent consumers in the electric cabinet that cannot be eliminated: such as frequency inverters, decentralized peripherals (switched magnetic valves, limit switches, etc.) or the Sinumerik controller itself. Besides further thermal losses in the system their static power consumption could neither be modeled nor compensated with the flywheel strategy.

According to the *ERMA* concept the total sum of the flywheel energy in the ERS together with the potential and kinetic energy of the machine must be kept at a constant energy level. The range of the computed *ENERGY* signal can be determined after the simulation. This has to be transformed into a range of flywheel speed according the inverse of the formula (1) in chapter 1.1. Right at the moment where the energy signal has its maximum, the flywheel must rotate at its lowest (idle) speed. Similarly, the upper flywheel speed-limit must correspond to the lowest energy state of the machine during the process. Using higher flywheel speed as needed would increase the friction losses, thus an optimum had to be found. The ideal flywheel speed range is also influenced by the maximum amount of energy that can be recov-

ered during the whole process. In case of typical fast pick-and-place tasks in a production line the flywheel of the ERS never stops, because it has relatively high power consumption upon startup accelerations. In case of the *ERMA* demonstrator the range of 5400-6800 rpm has been determined experimentally. Within this range of 1400 rpm 1300J of process energy can be stored theoretically.

As mentioned earlier, the simulation result file contains data about the original NC program lines, too. The splitting of the result file occurs at each moment the *LINE* signal changes. The example of Figure 7 yields six individual sections after the simulation. A new curve table definition is then created out of each section. It is not to forget that the flywheel must be controlled in advance. Therefore, prior writing the corresponding leader-follower axis values into the output curve table, the section of the energy signal is shifted 54 milliseconds forwards in time (towards prediction). This amount of time was determined empirically, after exhaustive testing with the real energy recuperation system. Figure 8 shows the software tool chain that is used in this work. A .NET solution “*IFF VNCK Manager*” has been created to modify an existing main NC program (.mpf) with the incorporation of curve tables, thus the NC controller is able to control the energy axis of the ERS transparently during the real executed process.

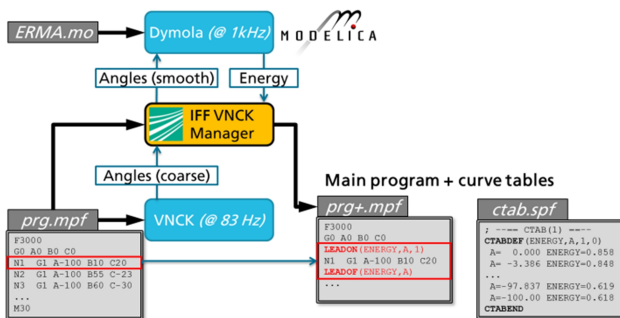


Figure 8: Software chain to create modified NC programs with ERS control capability

As there are usually multiple axes moving upon a single NC command, the leader axis of a curve table definition is chosen according to the biggest position difference being travelled. In the example of Figure 7 the role of the leader axis in the first three curve tables is A, B and C, respectively.

### 3 Summary

In this paper a new concept of including a flywheel-based energy recuperation system into production

machinery is presented. Based on a semi-automated workflow of creating a detailed, realistic mechatronic Modelica model of the system and coupling with a virtual controller, the movements to be executed can be assessed energetically in advance. This information can be used to control a mechanical energy storage unit earlier, overcoming its intrinsic flywheel inertia, which otherwise would hinder effective energy recuperation.

The demonstrator of the here presented *ERMA* project is located in the Virtual Development and Training Centre of the Fraunhofer IFF in Magdeburg, Germany. Using this methodology a 12% reduction of the machine’s energy consumption was achieved.

## 4 Acknowledgement

The here presented work has been funded by the German Federal Ministry of Education and Research within the Resource and Energy Efficiency Program under the project named “*ERMA - Energy Recuperation for Multi-Axis Machines*” (02PK2192).

## References

- [1] Juhasz, T.; Schmucker, U.: “Automatic Model Conversion to Modelica for Dymola-based Mechatronic Simulation”; Proc. of 6th International Modelica Conference, 3rd-4th March, 2008, Bielefeld, Germany, Vol. 2, pp. 719-726
- [2] Thielicke, R.; Adam, T.; Böhme, T.; Kennel, M.: “Entwicklung eines Werkzeuges zur automatisierten NC-Code-Generierung mit maschinennaher NC-Simulation” In: IFF-Wissenschaftstage, 2011 — ISBN 978-3-8396-0281-2
- [3] Mirtich, B.: “Fast and Accurate Computation of Polyhedral Mass Properties” Journal of Graphics Tools, Volume 1, Number 2, 1996
- [4] Schröder, D.: “Elektrische Antriebe - Regelung von Antriebssystemen”; Springer Verlag 2009; ISBN 978-3-540-89612-8
- [5] rosseta Technik GmbH - <http://www.rosseta.de/>
- [6] aradex AG - <http://www.aradex.de/en>
- [7] VINCENT - <http://www.iff.fraunhofer.de/de/geschaeftsbereiche/virtual-engineering/vincent.html>
- [8] AIT - [www.ait.ac.at](http://www.ait.ac.at)





# A Generalized Power-Based Modelica Library with Application to an Industrial Hydraulic Plant

Ilja Alkov, Robin Diekmann, Dirk Weidemann  
Institute of System Dynamics and Mechatronics  
University of Applied Sciences Bielefeld  
Wilhelm-Bertelsmann-Str. 10, 33602 Bielefeld, Germany  
{ilja.alkov, robin.diekmann, dirk.weidemann}@fh-bielefeld.de

## Abstract

This contribution presents a Modelica implementation of the generalized power-based modeling approach conforming to the bond graph methodology. The corresponding developed library BondGraph is discussed in detail. It allows graphical modeling according to the bond graph formalism, and contains common bond graph elements, as well as specific nonlinear elements, especially related to hydraulic effects. Furthermore, several composed models are provided, such as switching valves, pipes, cylinders, etc. A combination with blocks of the Modelica Standard Library is possible. The application of BondGraph to an industrial plant is described to demonstrate its capabilities.

**Keywords:** *power-based model, bond graph, library, Modelica, hydraulic*

## 1 Introduction

Since general purpose simulation environments enhance their capabilities, modelers attempt to take advantage of the offered possibilities completely. Hence, models become more comprehensive and cover the multidisciplinary of the considered systems. Therefore, domain specific modeling libraries are developed and offer possibilities to model the involved different physical effects separately as well as to interconnect them. The generalized power-based approach is an alternative efficient modeling formalism. Based on the generalized power definition, unified modeling elements are conceptualized applicable to the direct multidisciplinary modeling of complex systems. Consequently, the modeling procedure consists of the characterization of domain specific processes corresponding to their unified complements and interconnecting these complements to the model of complete sys-

tem according to its structure. For further advantages of generalized power-based modeling, see e.g. [1] and [2]. The Modelica language is particularly suitable for the development and representation of power-based models, primly because of the possibility of the acausal multiple signal connections definition.

The bond graph (BG) formalism provides a multidisciplinary, generalized power-based approach to the modeling and also graphical model representation of dynamic systems (cf. [3]). By the object-oriented nature, BG possess the according advantages. The graphical system representation by a BG can be translated automatically to a system of differential-algebraic equations, wherefore the implementation of the BG approach with an appropriate modeling language is necessary.

The Modelica language and the BG formalism are partly closely related modeling methodologies. Hence, an implementation of the BG formalism in Modelica is considered in this contribution. The proposed implementation attempts to take advantages of both modeling concepts. The developed open source library BondGraph is recommended to be used with Dymola 7 or later versions as this software supports completely included model definitions.

The introduced BondGraph library is available online at the official Modelica web page [4]. It contains common standard BG elements, as well as specific nonlinear elements. The included nonlinear models were designed especially for the modeling of hydraulic networks. For these, attention has been paid to their numerically stable computation. Further components provided by BondGraph include composed models of technical units, e.g. valves, and blocks for signal generation, e.g. to control valves.

A short introduction to the BG formalism is given in the Section 2. The implementation of a number of representative elements in the BondGraph library is

discussed in Section 3. In Section 3.1, the implementation of the standard BG elements, and in Section 3.2, the implementation of the hydraulic elements are presented. Section 4 introduces an industrial plant and the corresponding model developed using the BondGraph library. Several examples are provided to illustrate the described issues. A brief summary is given in Section 5. The used formula symbols are summarized in the Appendix.

## 2 Bond Graphs

Since its development, the BG approach has become a well-known technique for object-oriented graphical modeling (cf. [3]). This fact is based on the established definition of generalized power (cf. [5]) as the product of a conceptional effort  $e$  and a flow  $f$  variable:

$$P = f \cdot e. \quad (1)$$

This generalized specification provides the feasibility of multidisciplinary modeling with unified elements. The only prerequisite for this obviously advantageous modeling approach is an appropriate assignment of power variables i.e. effort and flow for considered domains.

For example, in the field of isothermal hydraulics, conventionally the pressure and the volumetric flow rate are assigned as the effort and the flow variable, respectively. A list of corresponding possible assignments for different energy domains is shown in Table 1.

Table 1: A selection of domains and corresponding conventional variable assignments in the bond graph formalism

Domain	Flow	Effort
hydraulic	volume flow rate	pressure
translational	velocity	force
rotational	angular velocity	torque
electrical	current	voltage
thermal	entropy flow rate	temperature

### 2.1 Bond Graph Elements

The basic set of BG elements consists of an effort source, a flow source, a capacitance, an inductance, a resistance, a transformer, and a gyrator, which are

given with corresponding constitutive equations in Table 2.

Table 2: Basic bond graph elements

Name	Element	Constitutive Equation
effort source	$Se$	$e = e_s$
flow source	$Sf$	$f = f_s$
capacitance	$C$	$e = \frac{1}{c} \int_{t_0}^t f(t^*) dt^* + e(t_0)$
inductance	$I$	$f = \frac{1}{i} \int_{t_0}^t e(t^*) dt^* + f(t_0)$
resistance	$R$	$e = r \cdot f$
transformer	$TF$	$e_1 = r_{if} \cdot e_2, f_1 = \frac{1}{r_{if}} \cdot f_2$
gyrator	$GY$	$e_1 = r_{if} \cdot f_2, f_1 = \frac{1}{r_{if}} \cdot e_2$

A significant extension of the modeling opportunities is offered by modulated elements. These are generalizations of linear elements where the proportionality coefficients are given by an external signal.

Two further multi-port elements referred to as junctions are defined in the bond graph methodology. These may represent conservation or equilibrium laws but also design constraints among the variables of the elements connected. In the graphical representation, the two power variables are carried by one eponymous bond connecting junctions and other BG elements, that is usually drawn as a half arrow. The flows of all elements connected to a 1-Junction are equal, whereas the sum of efforts carried by all incoming bonds equals the sum of efforts carried by all outgoing bonds. Again a 0-Junction is a dual element corresponding to the 1-Junction, hence for this the reversed relations of effort and flow are valid. For example, related to the hydraulic domain, all elements connected to a 1-Junction have the same flow as being connected in series, and all elements connected to a 0-Junction have the same effort as being connected in parallel.

Formally, a 1-Junction is described by the equations

$$\sum_{k=1}^n s_k \cdot e_k = 0, \quad s_k \in \{+1, -1\}, \quad (2)$$

$$f_1 = f_2 = \dots = f_n, \quad (3)$$

where  $n$  equals the number of elements connected to the junction. In the same manner, a 0-Junction is described by

$$\sum_{k=1}^n s_k \cdot f_k = 0, \quad s_k \in \{+1, -1\}, \quad (4)$$

$$e_1 = e_2 = \dots = e_n. \quad (5)$$

By convention,  $s_k = +1$  for all incoming bonds and  $s_k = -1$  for all outgoing bonds. Hence, the contributions to the constitutive equation of a junction are determined by the directions of the bonds in the graphical representation. Nevertheless, as can be seen by (2) and (4), formally, this is rather a property of the junction itself than of the connection. This fact is considered by the proposed implementation of the BG formalism in Modelica.

According to a specific domain, for each standard BG element a physical interpretation can be given. Considering a combination of linear elements, modulated elements, and sensors observing power variables, it is possible to describe a wide range of nonlinear models by case-related definitions of the external signal. Nevertheless, the definition of application-specific, and, where appropriate, nonlinear elements may lead to a convenient modeling process and ensures the clarity of the obtained models.

## 2.2 Causality

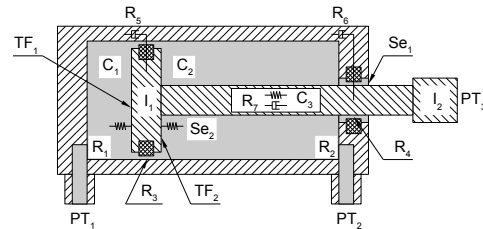
Within the BG formalism, bond connections are representations of information exchange between submodels. As two power variables are associated with each bond, two equations are obliged for their calculation and thus each end of the bond is responsible for one of them. Graphically, the flow determining end is marked by a short stroke. For example, energy storage elements are preferred to have integral causality, thus a capacitance preferably computes effort and an inductance preferably computes flow, whereas sources obviously determine their output variables. Several rules exist for causality assignment, e.g. listed in [3].

The graphical assignment of causality is useful if the equations describing the system behavior are directly deduced from the graphical BG representation. In some modeling tools as for example 20-sim (cf. [6]), the causality is fixed during the modeling procedure. However, it is not required to assign the causality during the modeling process using Modelica, as this is a subtask of the model compilation process, which is performed by suitable matching algorithms. For large models and generally for practical modeling, the manually or a priori fixed causality assignment is less reasonable. Therefore, manual causality assignment is not integrated in the presented BG implementation.

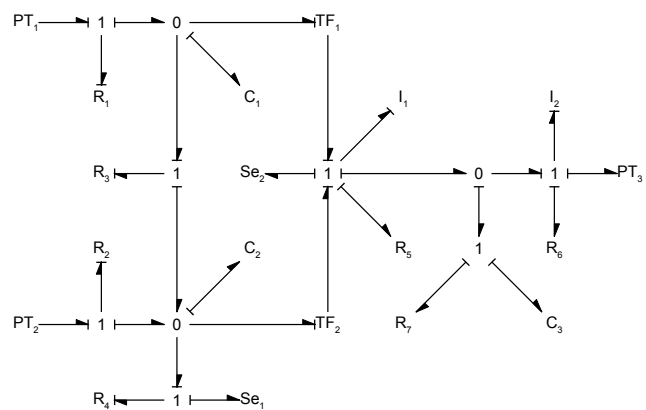
## 2.3 Example

An example of a simple physical system and its BG representation are shown in Fig. 1. The considered

system is a hydraulic cylinder driven by enforced pressures at hydraulic ports assigned with  $PT_1$  and  $PT_2$ . On the mechanical side, a load is executed represented by a force at the corresponding port  $PT_3$ . In this simple representation, basic BG elements are utilized exclusively, which may be less suitable for modeling with practical accuracy requirements.



(a) Iconic model of a hydraulic cylinder



(b) Classical bond graph representation

Figure 1: Simple bond graph example

## 3 BondGraph Library

### 3.1 Standard Bond Graph Elements

The set of standard bond graph elements is comprised of basic BG elements, their modulated complements, and junctions. Thus, these are included in the BondGraph library. Therefore, the corresponding constitutive equations and element interfaces are implemented in Modelica. The power variables are made accessible by acausal connectors, whereas for signals, input-output connectors are defined compatible to blocks of the Modelica Standard Library (MSL). For the energy storage elements, initial conditions are defined locally applying start attributes for differentiated variables. The element parameters are implemented according to the BG methodology. Extending this set, sensors are modeled as simple two port elements that do not affect the power variables. Via a

signal output interface, these elements provide power variables or their product i.e. power, corresponding time derivatives, or integrals. Again, the compatibility with blocks of the MSL is assured.

### 3.1.1 Bonds

According to the bond graph methodology, elements are interconnected by bonds. In contrast to the bonds, Modelica connections do not provide attributes for variable sign and for causality assignment, and their graphical representation. Therefore, if the bonds should be implemented closely to the original methodology, they would have to be objects of a class containing required properties. This approach would make the modeling procedure significantly more cumbersome.

The BondGraph library uses standard Modelica connections for the element interconnection introducing the connect equation directly or graphically. The graphical modeling technique is recommended here, as the BG formalism is a graphical modeling approach. Hence, an alternative method is used to determine the signs of power variables at a junction (cf. (2) and (4)). Therefore, junctions are equipped with a positive and a negative multiple port. These are indicated by a blue and a red circle, respectively. All elements connected to the positive port are considered with a positive sign ( $s_k = +1$ ) and all elements connected to the negative port are considered with a negative sign ( $s_k = -1$ ) in the sum of efforts for a 1-Junction, or in the sum of flows for a 0-Junction. In terms of the standard BG formalism, all incoming bonds are connected to the positive port and all outgoing bonds are connected to the negative port. This yields a clear graphical representation in compliance with (2) and (4) as discussed in Section 2.1.

### 3.1.2 Causality Assignment

As it has been discussed in Section 2.2, the causality assignment is not required by using Modelica. Furthermore, restrictions on algorithmic matching procedures may lead to less efficient model equations resolution. Hence, any regulations of this kind are avoided in the BondGraph library. Nevertheless, a graphical indication of the BG formalism specific causality assignment can be implemented in the animation layer extending the models of the released library by appropriate functionality. In this way, the standard BG causality assignment method might be visualized without eventual affecting of the equation resolution procedure.

### 3.1.3 BondGraph Example

The implementation of the example presented in Section 2.3 is shown in Fig. 2, now using the BondGraph library in Dymola. Here, the pressures and force imposed at the hydraulic and mechanical ports, respectively, are implemented utilizing effort sources and look-up tables from the MSL.

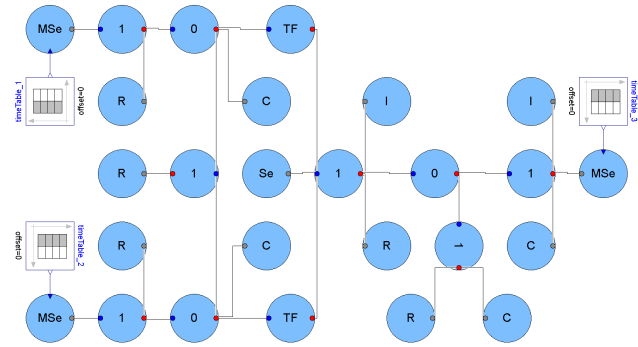


Figure 2: Simple bond graph example implemented using BondGraph

## 3.2 Hydraulic Elements and Effects

Hydraulic processes are subprocesses of fluid-mechanical power exchange. For the modeled processes, this is considered transient and spatially one-dimensional along a main flow direction. Models of the following hydraulic effects are included in the BondGraph library: capacitive energy storage (*HC* and *MHC*), inductive energy storage (*HI*), dissipation (*HR*, *HRL*, *HRT*, and *MHRT*), and processes with source characteristics (*HSe acc* and *HSe ind*). In the hydraulic domain, the power variables  $e$  effort and  $f$  flow are pressure and volume flow, respectively.

### 3.2.1 Working Fluid Properties

For the modeling of these hydraulic processes, the properties of the working fluid are of essential interest, since these may vary significantly within the working range. The *hprop* model of the BondGraph library provides a computation of the hydraulic fluid properties density and viscosity depending on the fluid temperature and the absolute pressure. By default, it is parameterized concerning the working fluid HLP ISO VG 32 as specified by the German industry norm DIN 54524, part 2. The introduced model is valid for the temperature range of [288, 363] K and the feasible pressure lies in a wide range of [1, 10<sup>8</sup>] Pa. The properties computed by the *hprop* model play an important role for

other hydraulic effects and therefore it is contained in the models of hydraulic processes.

For each process, a decisive pressure is defined by the arithmetic mean of the pressures before and after the considered process. This definition assures the independence of the flow direction and is of sufficient accuracy in most cases. Different to the standard bond graph convention, hydraulic processes are implemented as two-port elements in the BondGraph library where the mean values of the pressures at the ports, and also the fluid properties as determined by  $hprop$ , are automatically considered.

The dependence of fluid properties upon other conditions besides pressure and temperature is not modeled. The working fluid is assumed to be a homogeneous mixture of hydraulic oil and air, whereby the relative mass parts in the mixture are assumed to be time and state invariant.

**Density.** The density of the hydraulic oil is described by the equation

$$\rho_{oil} = \rho_{oil,ref} \cdot \exp(\kappa_{oil} \cdot (p - p_{ref}) - \gamma_{oil} \cdot (T - T_{ref})), \quad (6)$$

considering the compressibility and thermal expansion of the oil by the parameters  $\kappa_{oil}$  and  $\gamma_{oil}$ , respectively, whereby the density of air is modeled by the ideal gas law

$$\rho_{air} = \rho_{air,ref} \cdot \frac{p}{p_{ref}} \cdot \frac{T_{ref}}{T}. \quad (7)$$

These densities are then used to obtain the the density of the mixture by

$$\rho = \left( \frac{\mu_{oil}}{\rho_{oil}} + \frac{\mu_{air}}{\rho_{air}} \right)^{-1}, \quad (8)$$

which may be parameterized by the relative mass parts of the hydraulic oil and air.

It is also possible to obtain the compensated compressibility modulus of the working fluid from this model (cf. [7]). Thus it was verified with the conversed available data for the compensated compressibility modulus.

**Viscosity.** The viscosity of the hydraulic oil is modeled by the Roelands relation (cf. [8])

$$\eta_{oil} = \eta_{oil,ref} \cdot \exp \left( \ln \left( \frac{\eta_{oil,ref}}{6.315 \cdot 10^{-5} \text{Pa} \cdot \text{s}} \right) \cdot \psi \right), \quad (9)$$

with

$$\psi = -1 + \left( 1 + \frac{p - p_{ref}}{1.96 \cdot 10^8 \text{Pa}} \right)^\zeta \cdot \left( \frac{T - 138\text{K}}{T_{ref} - 138\text{K}} \right)^\xi. \quad (10)$$

The viscosity of air is described by the own approximation

$$\eta_{air} = \eta_{air,ref} \cdot \frac{\theta(p, T)}{\theta(p_{ref}, T_{ref})}, \quad (11)$$

with

$$\theta(p, T) = p^{0.01} \cdot T^{0.75} + 0.132 \cdot p \cdot T^{-2}. \quad (12)$$

Data used for the approximation is available in [9].

For the calculation of the viscosity of the mixture, relative volume parts of the components are obtained by

$$\phi_{oil} = \frac{\frac{\mu_{oil}}{\rho_{oil}}}{\frac{\mu_{oil}}{\rho_{oil}} + \frac{\mu_{air}}{\rho_{air}}} \quad (13)$$

and

$$\phi_{air} = \frac{\frac{\mu_{air}}{\rho_{air}}}{\frac{\mu_{oil}}{\rho_{oil}} + \frac{\mu_{air}}{\rho_{air}}}. \quad (14)$$

With reference to it's physical characteristic, the viscosity of the homogeneous mixture of hydraulic oil and air is then described by

$$\eta = \frac{\eta_{oil} \cdot \phi_{oil}^{2/3} + \eta_{air} \cdot \phi_{air}^{2/3}}{\phi_{oil}^{2/3} + \phi_{air}^{2/3}}. \quad (15)$$

### 3.2.2 Hydraulic Capacitance

The capacitive storage capability has a significant effect on the dynamic behavior of hydraulic systems. This property results from the elasticity of the considered hydraulic component and consequently from the pressure dependent variability of the enclosed fluid volume in the component. At low pressure, the compressibility of the working fluid has a considerable effect on the capacitive storage capability. Both effects are considered by the hydraulic capacitance model  $HC$  and mathematically described by

$$f = \frac{1}{\rho} \cdot \frac{d}{dt}(\rho \cdot v), \quad (16)$$

with

$$v = v_0 \cdot \exp(c_v \cdot (e - e_0)), \quad (17)$$

where  $e$  is the pressure,  $e_0$  is the initial pressure, and  $f$  is the volume flow difference taken by the component from the main volume flow.

Whereas the initial volume  $v_0$  is a parameter in the  $HC$  model, it can be adjusted by an external signal in the  $MHC$  model. This extends the modeling opportunities and e.g. allows the consideration of components with moving parts as for instance cylinders.

### 3.2.3 Hydraulic Inductance

From a power-based point of view, the inductive energy storage is the dual process to the capacitive storage and it mainly results from the inertia of the working fluid. The relation between the power variables in the inductive storage element is obtained from the principle of linear momentum applied on the enclosed working fluid in the component by

$$e = \frac{d}{dt} \left( \frac{l \cdot \rho}{A} \cdot f \right), \quad (18)$$

where  $e$  is the pressure difference across the component.

### 3.2.4 Hydraulic Dissipation

Dissipative hydraulic processes are highly nonlinear and change their characteristic dependent on the transient flow conditions (cf. [10]). As the hydraulic resistance is described using different models according to the particular effect, several elements are provided by the BondGraph library: strait pipe resistance ( $HR$ ), resistance of a pipe fitting with laminar flow ( $HRL$ ), resistance of a pipe fitting with turbulent flow ( $HRT$ ), and switchable hydraulic resistances ( $MHRT$ ) are distinguished.

**Strait pipe resistance.** The dissipative resistance of strait pipes is extensively studied in the literature (cf. e.g. [11], [12], [13]). The model proposed in this contribution (given by the following set of equations (19)-(25)) is based on the Darcy equation (equation (19), cf. [14]). The pipe friction factor for the entire range of the Reynolds number is obtained with an explicit, continuous approximation of the pipe friction factor equation for laminar flow and of the Colebrook-White equation (cf. [15]).

As can be seen by Fig. 3, the resulting pipe friction factor function  $\lambda(Re)$  (cf. equation (22)) is continuous as well as continuously differentiable, it does not diverge, and also does not include distinctions between different cases (if-then constructs), which yields stable computability. Due to the explicit form of equation (24), our approach also avoids the computationally expensive solution of the originally proposed implicit equation for turbulent flows.

$$e = \lambda \cdot \frac{l \cdot \rho}{2 \cdot d_h \cdot A^2} \cdot |f| \cdot f \quad (19)$$

$$Re = \frac{d_h \cdot \rho \cdot |f|}{A \cdot \eta} \quad (20)$$

$$Re_t = \frac{2 \cdot 6.9}{1 - \left(\frac{r_h}{3.7}\right)^{1.11}} \quad (21)$$

$$\lambda = \frac{\lambda_l}{1 + \exp\left(\frac{Re - Re_{crit}}{0.228 \cdot Re_{range}}\right)} + \frac{\lambda_t}{1 + \exp\left(\frac{Re - Re_{crit}}{-0.228 \cdot Re_{range}}\right)} \quad (22)$$

$$\lambda_l = \frac{64}{Re + Re_{min} \cdot \left(1 - \tanh\left(\frac{Re}{Re_{min}}\right)\right)} \quad (23)$$

$$\lambda_t = \left(\frac{-1.8}{\ln(10)} \cdot \ln\left(\left(\frac{r_h}{3.7}\right)^{1.11} + \chi\right)\right)^{-2} \quad (24)$$

$$\chi = \frac{6.9}{Re + Re_t \cdot \left(1 - \tanh\left(\frac{Re}{Re_t}\right)\right)} \quad (25)$$

In this equation set,  $e$  is the pressure difference across the component. To obtain numerical stability using sigmoidal functions, the following parameters are utilized:  $Re_{crit}$  is the Reynolds number value for the laminar-turbulent flow transition,  $Re_{range}$  is the Reynolds number range for the laminar-turbulent flow transition and  $Re_{min}$  is the Reynolds number value for the zero-flow crossing.

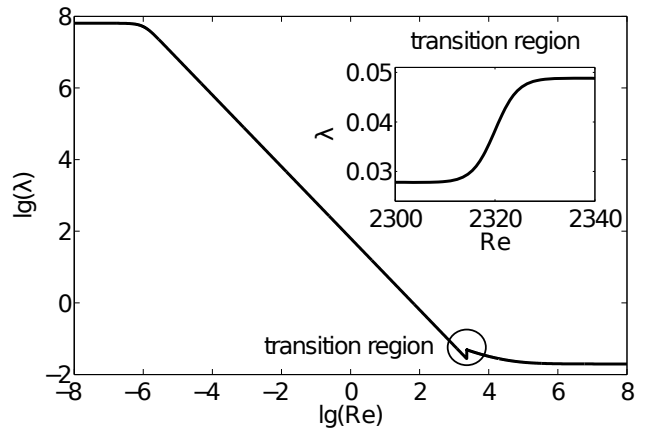


Figure 3: Proposed approximation of pipe friction factor  $\lambda$  according to equation (22). The inset shows the laminar-turbulent transition region in detail with typical values for  $Re_{crit}$  and  $Re_{range}$ .

**Resistance of pipe fittings.** The effect of dissipation due to hydraulic fittings is available by the models  $HRL$  and  $HRT$ . These may be used in case of hydraulic apertures, throttles, bendings, cross-section changes, and similar components. Resistances in these components (cf. [7]) can be described by taking their flow characteristic into account. In case of laminar flow, the viscosity of the fluid strongly affects the resistance which is expressed by the equation

$$e = \frac{e_{ref} \cdot \eta}{\eta_{ref} \cdot f_{ref}^\gamma} \cdot |f|^{\gamma-1} \cdot f. \quad (26)$$

Contrariwise, in case of turbulent flow, the density of the fluid has to be considered:

$$e = \frac{e_{ref} \cdot \rho}{\rho_{ref} \cdot f_{ref}^\gamma} \cdot |f|^{\gamma-1} \cdot f. \quad (27)$$

In both cases,  $e$  is the pressure drop across the component. For both laminar and turbulent resistances, often flow and pressure drop are not in linear relation to each other, but their relation may be described by a characteristic flow exponent  $\gamma$ , e.g.  $\gamma = 2$  if the flow is proportional to the square root of the effort. However, it usually depends on the geometry of the fitting and may therefore be adjusted by a parameter of the model.

**Switchable hydraulic resistance.** Hydraulic valves may be interpreted as switchable resistances and thus be modeled by switchable hydraulic elements. The *MHRT* element represents a signal controlled hydraulic resistance with turbulent characteristic where an input signal allows to control the state of the hydraulic resistance. An input value of 1 corresponds to a complete opened valve, whereas a value of 0 corresponds to a completely closed valve. To generate appropriate control inputs, several blocks are also provided by the BondGraph library. In this way the physical multi-domain nature of valves may be taken into account: though valves affect the hydraulic resistance, they are controlled by mechanical systems which are actuated either manually or by further technical units, e.g. a solenoid actuators. Therefore, array types of power-based models can be efficiently used for the modeling of hydraulic valves (cf. [16]).

### 3.2.5 Processes with Source-Characteristic

**Height change.** In the fluid-mechanical power exchange processes, two subprocesses with effort-source-characteristic are often involved. One of them is the pressure change  $e$  due to fluid displacement in the earth gravitation field, therefore

$$e = \rho \cdot g \cdot d, \quad (28)$$

which is considered by the *MSe acc* model.

**Cross section area change.** The second subprocess is covered by the *MSe ind* element and models the pressure change  $e$  due to a cross sectional area change in hydraulic lines, thus

$$e = \frac{\rho}{2} \cdot |f|^2 \cdot \left( \frac{1}{A_1^2} - \frac{1}{A_2^2} \right). \quad (29)$$

### 3.2.6 Consideration of Nonlinearity

**Causalized elements.** As pointed out in Section 2.2, causality does not have to be assigned in general by using Modelica. Nevertheless, when using nonlinear elements e.g. *HR*, *HRT*, *HRL*, or *MHRT*, it is recommended that the designer observes whether the element has effort-out or flow-out causality in the actual model. Accordingly, the activation of an appropriate formulation of the element equations should be controlled by a parameter. In this way, unnecessary numerical inversion of the nonlinear element equations is avoidable. Hence, the solution time of the model can be shortened and the solution accuracy and stability can be increased.

For instance, in (30) and (31), both causalized forms of the turbulent resistance are given (cf. equation (27)).

$$e = \frac{e_{ref} \cdot \rho}{\rho_{ref} \cdot f_{ref}^\gamma} \cdot |f|^{\gamma-1} \cdot f \quad (30)$$

$$f = \left( \frac{e_{ref} \cdot \rho}{\rho_{ref} \cdot f_{ref}^\gamma} \right)^{\frac{-1}{\gamma}} \cdot |e|^{\frac{1}{\gamma}-1} \cdot e \quad (31)$$

In (32), a form corresponding to the linear resistance element is obtained. The state dependent resistance coefficient  $r$  is separated. Different formulations for the resistance coefficient are then used dependent on the  $par_{caus}$  parameter.

$$e = r \cdot f, \quad (32)$$

with

$$r = \begin{cases} \frac{e_{ref} \cdot \rho}{\rho_{ref} \cdot f_{ref}^\gamma} \cdot |f|^{\gamma-1} \cdot f & \text{if } par_{caus} = 1, \\ \left( \frac{e_{ref} \cdot \rho}{\rho_{ref} \cdot f_{ref}^\gamma} \right)^{\frac{1}{\gamma}} \cdot |e|^{1-\frac{1}{\gamma}} \cdot e & \text{if } par_{caus} = 2. \end{cases} \quad (33)$$

**Variation of the proportionality factors.** Corresponding reformulations and separations of the proportionality factors are often possible and convenient for nonlinear elements with other characteristics. As these coefficients in such reformulated relations are in general time and state variant, their absolute values can attain low values or zero. The same has to be considered for the reciprocal of the absolute value. In these cases, structural changes arise in the interconnected model equations and a numerical solution is either not possible or becomes unstable. For instance, this might result in a non-decaying oscillation of the power variables where the amplitude and particularly the frequency of the oscillation depend on the chosen

solver algorithm and solution tolerance. In this case, an effective and simple modification of the model is the limitation of the value of the proportionality coefficient. The mentioned numerically induced oscillations are avoidable completely by this model extension.

### 3.3 Comparison Remark to BondLib

BondLib is a former successful implementation attempt of the bond graph formalism in the Modelica language presented in [17] awarded as the best free Modelica library in the framework of the Modelica Conference 2005. BondGraph is a conceptually different modeling library, intended rather for pragmatic practical modeling using bond graph methodology then for educational purpose. In contrast to BondLib, BondGraph does not include elements closely corresponding to bonds. Instead, Modelica standard connections are utilized for the interconnection of BG elements. Consequently, the assignment of the variable signs is implemented as a property of junctions instead of using the directional property of bonds. In this way, BondGraph utilizes Modelica more efficiently slightly differing from the graphical representation defined in the BG methodology. Since bonds as elements are excluded, in comparison to a model set up with BondLib, an equivalent model set up with BondGraph involves significantly less elements. Furthermore, the graphical clarity for large models can be maintained more straightforward. The junction implementation in BondGraph permits connection of elements of unlimited number, whereas in the BondLib, junctions with different fixed numbers of ports are available. Hence, model modifications and expansions are performed more efficiently utilizing BondGraph, since originally used junctions may be retained. From this point of view, the BondGraph junction implementation is closer to the BG methodology. The BondLib library offers bonds with fixed causality. As described before, BondGraph avoids any regulations of this kind well motivated following more closely the Modelica approach. Concluding this short comparison, we suggest BondGraph for practical modeling, whereas we recognize that BondLib offers several properties very important for bond graph beginners and educational purpose.

## 4 Industrial Hydraulic Plant

The discussed library is utilized for the modeling of an industrial hydraulic plant. The considered plant

is a riveting system with a hydraulic power transmission. The system can be structured into the following principal units: a hydraulic power supply unit containing an inductance motor and a pump, a valve block, a hydraulic-mechanical actuator, hydraulic pipes interconnecting these units, and a sequence control. Hence, the model to be developed should cover a comprehensive hydraulic network. Figure 4 shows the model top layer.

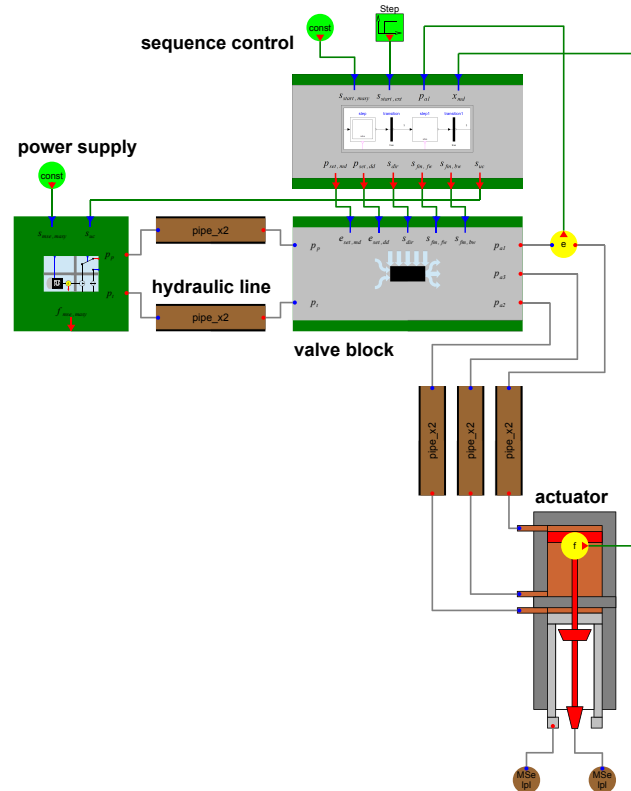


Figure 4: Model of the considered riveting system

The submodels representing principal units are modeled using the BondGraph library and the MSL. A plain example for this model set-up is the model of a hydraulic pipe consisting of hydraulic elements exclusively. A series of hydraulic dissipative and inductive elements separated by hydraulic capacitive elements may be utilized for the description of the pipe behavior. The series starts and ends with turbulent resistances representing resistances of the fittings. The inner part is the repeating of series consisting of a straight pipe resistance and a hydraulic inductance again separated by capacitive elements. This repeating incorporates spatial discretization of the pipe along the main flow direction. Consequently it should be undertaken an appropriate number of times to cover relevant system eigenvalues by the model. An example of pipe



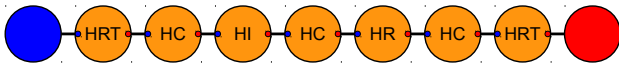


Figure 5: Model of the hydraulic pipe

model with one inductive storage is given in Fig. 5.

The model of the complete plant is a higher index differential-algebraic system consisting of 4923 equations with 106 continuous time states. The operation cycle of the plant of 1.7 s is simulated. Figure 6 shows the normalized displacement  $x$  of the actuator effector, the mechanical energy  $E$  supplied by the inductance motor in the power supply unit, and the pressure  $p$  in the hydraulic circuit.

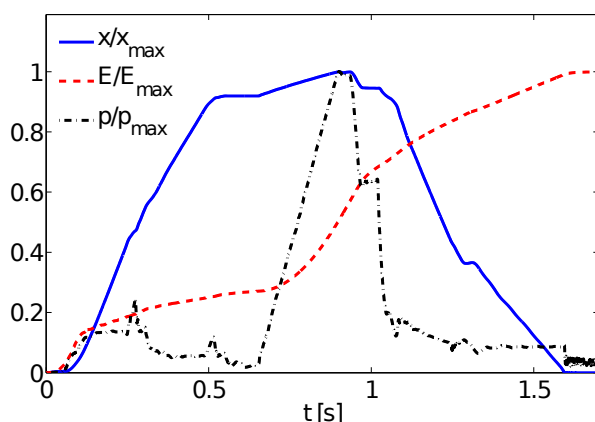


Figure 6: Simulation results. Plotted are the displacement  $x$  of the actuator, the mechanical energy  $E$  of the inductance motor, and the pressure  $p$  supplied by the pump.

The model parameters are identified by an optimization approach with reference to measurements obtained at a demonstrator plant. Hence, the model is verified against comprehensive experimental data. Furthermore, simulation based optimization of the plant is succeeded with the objective function defined as the consumed energy for an operation cycle (cf. [18]).

## 5 Conclusion

The bond graph methodology is a generalized power-based modeling approach. Hence, it is particularly advantageous for modeling of multidisciplinary systems. Therefore, BondGraph, the Modelica implementation of this approach enables applicants to take advantage of both, the flexible Modelica language and the well structured generalized power-based bond graph formalism. Furthermore, utilizing Modelica as the im-

plementation language, efficient general purpose modeling and simulation environments are made available for the bond graph applicants. Besides the development of BondGraph, a practical application of the library is conducted. Thereby, an industrial hydraulic plant is modeled, simulated, and optimized utilizing the developed modeling library.

## Acknowledgments

This elaboration was co-funded by the European Union (European Funds for Regional Development) and North Rhine-Westphalia, Germany (Federal State of Germany).

## References

- [1] D. Jeltsema and J. M. A. Scherpen, "Multidomain modeling of nonlinear networks and systems," *IEEE Control Systems Magazine*, vol. 29, pp. 28–59, 2009.
- [2] R. A. Layton, *Principles of Analytical System Dynamics*. Springer, 1998.
- [3] W. Borutzky, *Bond Graph Methodology*. Springer, 2010.
- [4] I. Alkov and R. Diekmann. (2013) BondGraph library. [Online]. Available: <https://www.modelica.org/libraries>
- [5] H. M. Paynter, *Analysis and Design of Engineering Systems*. M.I.T. Press, 1961.
- [6] J. F. Broenink, "20-sim software for hierarchical bond-graph/block-diagram models," *Simulation Practice and Theory*, vol. 7, no. 5-6, pp. 481–492, 1999.
- [7] H. Murrenhoff, *Grundlagen der Fluidtechnik - Teil 1: Hydraulik*. Shaker, 2012.
- [8] C. J. A. Roelands, J. C. Vlugter, and H. I. Waterman, "The viscosity-temperature-pressure relationship of lubricating oils and its correlation with chemical constitution," *ASME Journal of Basic Engineering*, vol. 11, pp. 601–611, 1963.
- [9] J. H. Spurk and N. Aksel, *Strömungslehre, Einführung in die Theorie der Strömungen*. Springer, 2010.

[10] R. Etlender, “Modellierung und Simulation der Wellenausbreitung in flexiblen hydraulischen Leitungen,” Ph.D. dissertation, Universität Stuttgart, 2012.

[11] J. Nikuradse, “Strömungsgesetze in rauhen Rohren,” *V.D.I. Forschungsheft*, vol. 361, pp. 1–22, 1933.

[12] C. F. Colebrook, “Turbulent flow in pipes, with particular reference to the transition region between the smooth and rough pipe laws,” *Journal of Institution of Civil Engineers*, vol. 11, no. 4, pp. 133–156, 1939.

[13] L. F. Moody, “Friction factors for pipe flow,” *Transactions of the ASME*, vol. 66, pp. 671–684, 1944.

[14] H. Darcy, *Les Fontaines Publiques de la Ville de Dijon*. Dalmont, 1856.

[15] S. E. Haaland, “Simple and explicit formulas for the friction factor in turbulent pipe flow,” *Transactions of the ASME, Journal of Fluids Engineering*, vol. 105, pp. 89–90, 1983.

[16] I. Alkov, “Verallgemeinertes Modell hydraulischer Ventile,” 2012, FLUIDON Conference 2012.

[17] F. E. Cellier and A. Nebot, “The Modelica bond graph library,” *Proceedings of the 4th International Modelica Conference*, pp. 57–65, 2005.

[18] J. Heinze, R. Diekmann, I. Alkov, and D. Weidemann, “Model-based energy optimization of assembly systems,” 2013, 35th IAT Colloquium of Automation.

$f$	flow
$f_0$	initial flow
$f_k$	flow at port $k$
$f_{ref}$	reference flow
$f_s$	source flow
$g$	gravitational acceleration
$i$	inductance coefficient
$k$	port counter
$l$	extend of enclosed fluid volume
$n$	port number
$P$	power
$p$	pressure
$p_{ref}$	reference pressure
$par_{caus}$	parameter for causalization
$Re$	Reynolds number
$Re_{crit}$	Reynolds number for laminar-turbulent transition
$Re_{min}$	Reynolds number for zero-flow crossing
$Re_{range}$	Reynolds number range for laminar-turbulent transition
$Re_t$	Reynolds number limiting parameter for turbulent region
$r$	resistance coefficient
$r_h$	relative hydraulic roughness
$r_{tf}$	power transformation coefficient
$s_k$	sign parameter of port $k$
$T$	temperature
$T_{ref}$	reference temperature
$t, t^*$	time
$t_0$	initial time
$v$	enclosed fluid volume
$v_0$	initially enclosed fluid volume
$x$	displacement
$x_{max}$	maximal displacement value
$\gamma_{oil}$	thermal expansion factor of oil
$\gamma$	volume flow exponent
$\zeta$	parameter of Roelands relation
$\eta$	viscosity of working fluid
$\eta_{air}$	viscosity of air
$\eta_{air,ref}$	reference viscosity of air
$\eta_{oil}$	viscosity of hydraulic oil
$\eta_{oil,ref}$	reference viscosity of hydraulic oil
$\eta_{ref}$	reference viscosity of fluid
$\theta$	auxiliary function in equation for air viscosity
$\kappa_{oil}$	compressibility factor of oil
$\lambda$	friction factor
$\lambda_l$	friction factor, laminar region
$\lambda_t$	friction factor, turbulent region
$\mu_{oil}$	relative mass part of hydraulic oil
$\mu_{air}$	relative mass part of air
$\xi$	parameter of Roelands relation
$\rho$	density of working fluid
$\rho_{air}$	density of air
$\rho_{air,ref}$	reference density of air
$\rho_{oil}$	density of hydraulic oil
$\rho_{oil,ref}$	reference density of hydraulic oil
$\rho_{ref}$	reference density of fluid
$\phi_{air}$	relative volume part of air
$\phi_{oil}$	relative volume part of hydraulic oil
$\chi$	auxiliary function in pipe friction factor equation
$\psi$	auxiliary function in Roelands relation

## Appendix

### List of Formula Symbols

$A$	cross section area of enclosed fluid volume
$A_1$	cross section area before change
$A_2$	cross section area behind change
$c$	capacitance coefficient
$c_v$	volumetric compliance factor
$d$	height difference
$d_h$	hydraulic diameter
$E_{mech}$	mechanical energy
$E_{mech,max}$	maximal mechanical energy value
$e$	effort
$e_0$	initial effort
$e_k$	effort at port $k$
$e_{ref}$	reference effort
$e_s$	source effort

# Physical Design of Hydraulic Valves in Modelica

Sureshkumar Chandrasekar   Hubertus Tummescheit  
Modelon Inc., Hartford, CT, USA  
chandrasekar@modelon.com   hubertus.tummescheit@modelon.com

## Abstract

This paper focuses on the physical design of hydraulic relief and servo valves and its applications. Specifically, this paper serves to illustrate how the physical design parameters of hydraulic components can be incorporated into system modeling and their effect on the system dynamics and stability characteristics. Detailed physical models of a relief valve and a servo valve developed using the Hydraulics Library<sup>®</sup> are discussed in this paper with particular emphasis on the effect of design parameters on the stability characteristics. A simple design of experiment (DoE) to illustrate robust design methods for hydraulic system design is also shown with the use of the FMI Toolbox (FMIT) for MATLAB<sup>®</sup>. Furthermore with the help of these two valve models, we seek to bring to the attention of the community, a limitation in open loop controls analysis in an acausal modeling environment where the feedback loops are embedded in the physics of the model.

*Keywords: Hydraulic Valves; Hydraulic Component Design; Mechanical feedback; Stability Analysis; Hydraulics Library; FMIT*

## 1 Introduction

Hydraulic control valves use mechanical motion to control or regulate fluid power in a hydraulic circuit. These valves serve as the interface between the mechanics and fluid dynamics in hydraulic systems and their performance characteristics are therefore critical to the safe and optimal operation of the systems in which they are employed [1]. Relief and servo valves are among critical components used in a wide variety of hydraulic systems and are respectively operated to regulate supply pressure to the load and convert fluid power to mechanical actuation.

In this paper, we discuss detailed Modelica models of a pressure relief valve and a servo valve and include the physical design parameters of these valves

and study the effect of these on the stability characteristics.

For the modeling of these valves, the newly developed Elements package of the Hydraulics Library is used. The Hydraulics Library is a commercial Modelica library from Modelon AB that enables modeling of fluid power systems in Modelica. The Elements package is developed to incorporate hydraulic component sizing into the design and modeling process. The library provides a comprehensive set of components typically used in hydraulic system modeling and provides the flexibility to build complex models from the basic building blocks available from the library. The Hydraulics Library with the Elements package is available for use with both Dymola and MapleSim<sup>®</sup>.

Due to its application in multiple engineering domains, the library provides a set of basic building blocks of components from which a more detailed model of a hydraulic system can be constructed. The elements package for example, provides models of a spool and a piston, which are connected through mechanical and hydraulic port connectors in any number of ways to construct detailed models of relief and directional control valves.

In this paper, we present detailed models of a Pressure Relief Valve and an Electro-Hydraulic Servo Valve (EHSV) constructed using components from the Hydraulics Library. Further, a Functional Mock-up Unit (FMU) of these models is imported into MATLAB<sup>®</sup> using the Functional Mock-up Interface Toolbox<sup>1</sup> for MATLAB [5] and linearized to study their stability characteristics and demonstrate robust design techniques. The two examples of the valves, apart from demonstrating the physical design of hydraulic components, also showcase a limitation in an acausal modeling environment where feedback loops embedded within the physics of the model need to be broken for analysis purposes. In controls analysis, such systems can only be linearized by manually

<sup>1</sup>FMIT for MATLAB is a commercial FMI tool from Modelon AB for the integration of physical models in MATLAB/Simulink.

modifying the physical equations. In our opinion, this is not an elegant solution and we illustrate this point through the two examples of the valves.

## 2 Hydraulic Elements Package

The Elements package of the Hydraulics Library provides an easy to use interface to model both the mechanics and hydraulic characteristics of hydraulic systems. Figure 1 illustrates the elements package. The subsequent subsections briefly discuss the various components of the elements package and their features.

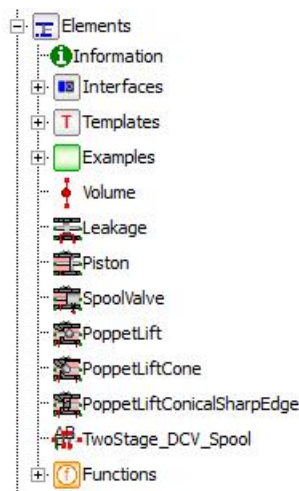


Figure 1: The Elements package of the Hydraulic Library

### 2.1 Connecting port interfaces

Three types of connecting ports are used in the components of the Elements package and these are illustrated in Figure 2.

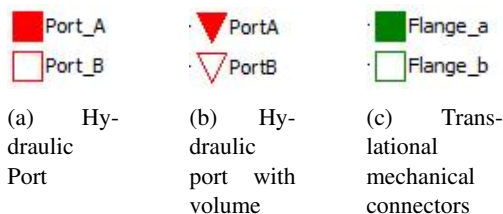


Figure 2: Connecting port interfaces used in the Hydraulics Library

The hydraulic port connectors in Figure 2(a) are the default connectors used throughout the hydraulics library with pressure and mass flow rate as the potential

and flow variables respectively. The connectors in Figure 2(b) are similar to those of Figure 2(a) except for the presence of the auxiliary volume variable. This port is used to connect the variable volume element with components where the volume change arises. The mechanical connectors in Figure 2(c) are the translational mechanical connectors from the Modelica Standard library with displacement and force as the potential and flow variables respectively.

### 2.2 Volumes

The Elements package provides two volume components that differs from the standard lumped volume component in the Volumes package of the library in that the volume is computed outside of the component and determined from the volume variable of the hydraulic port with volume. The first volume component (Figure 3(a)) represents a variable lumped volume where the pressure in the volume is calculated from the following continuity equation:

$$V \frac{dp}{dt} = \frac{\beta}{\rho} \frac{dm}{dt} - \beta \frac{dV}{dt} \tag{1}$$

where  $p$  is the pressure in the volume,  $V$  is the volume computed from the outside of the component and used through the port,  $\rho, \beta$  are the density and bulk modulus of the hydraulic fluid computed at pressure  $p$  and  $m$  is the mass flow through the volume component.

The second volume component (Figure 3(b)) does not include any pressure dynamics and is solely intended for connections between the standard hydraulic port and the port with volume.

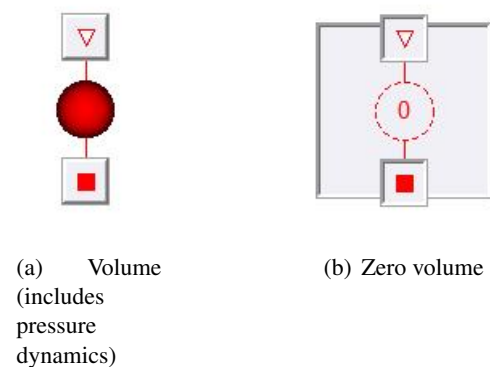


Figure 3: Hydraulic volumes used in the Hydraulics Library

## 2.3 Valve Components

The basic building blocks of valve components provided in the Elements package are for leakage, piston, spool and poppet lift and these are illustrated in Figure 4.

- The leakage component (Figure 4(a)) models laminar flow of fluid leakage due to radial clearance, underlap or overlap and the leakage path length is assumed to be a constant. Variable length leakage is expected to be included in a future release of the library.
- The piston (Figure 4(b)) component represents a hydraulic piston with an open chamber ending at both sides. The chamber volume is computed as a function of the displacement due to the net force on the piston head. The pressure and mass flow rate of the fluid into/out of the piston is computed as a function of this change in chamber volume.
- The spool valve (Figure 4(c)) component models the spool and sleeve of a 2-way single land spool valve. The fluid flow through the spool is a function of the spool displacement modeled as a variable area orifice. The flow area of this orifice is computed from the geometry of the valve opening and this option is parameterized with 4 options: linear, quadratic, circular or custom (tabular user input). The spool valve also includes the effect of flow forces computed from the spool displacement and pressure drop across the valve. The valve openings are modeled as orifices and the expressions are regularized for numerical stability in the orifice equations in these components.
- Three variations of a poppet valve (Figure 4(d)-4(f)) are also provided in the library, each differing in the geometry of the poppet lift and seat. Like the spool valve, each of these poppet valves includes the effect of flow forces and regularized expressions for numerical robustness.

## 3 Modeling and Open-Loop Stability Analysis of a Hydraulic Pressure Relief Valve

Hydraulic relief valves act to regulate pressure in a hydraulic system where the pressure to be controlled

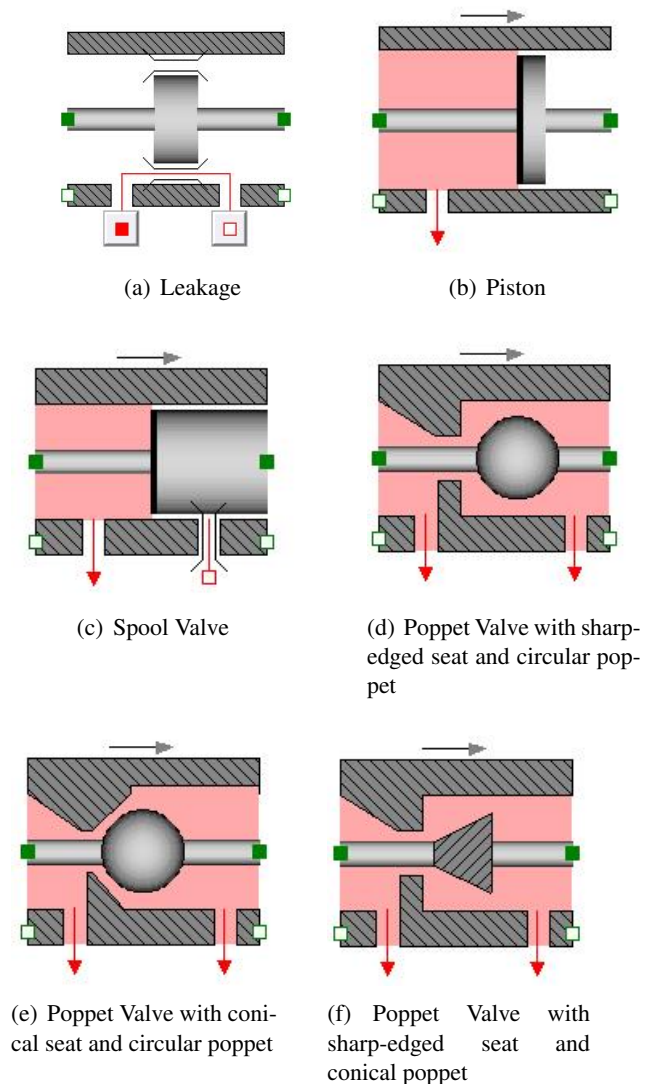


Figure 4: Building blocks to model hydraulic valves

is sensed in the valve end area and compared with a spring force setting [1]. In this work, the pressure relief valve described in [1] is modeled as a 3-way 2-land spool valve in Dymola using components from the Modelica Standard Library and the Hydraulics Library and the schematic of the valve is shown in Figure 5.

The spring with spring constant  $K_s$ , is preloaded with a force of  $F_0$ . This preloading sets the pressure  $P_c$ , in the spool valve chamber such that the valve is open with a displacement of  $x$  at steady state. As the supply pressure to the load increases, the spring is compressed to open the spool valve to allow more flow through the valve and into the tank maintained at atmospheric pressure. This relieves the excess pressure in the supply line. The opening of the valve is restricted between the geometric constraints of the spool and the supply

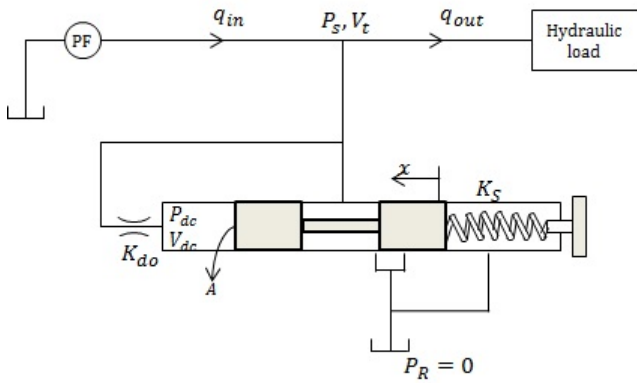


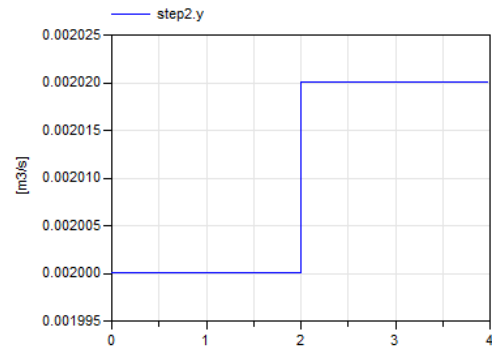
Figure 5: Pressure Relief Valve [1]

pressure determines the level of opening of the valve. The Modelica model of the relief valve is shown in Figure 15.

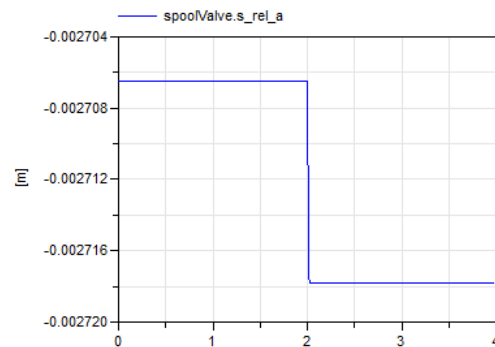
The geometric parameters of the valve were obtained from data sheets from Eaton Corp. [3] and the procedure for the derivation of valve parameters from data sheets is described in [4]. These parameters and the boundary conditions of the system are tabulated in Table 1. In Figure 15, the pistons are identical with the exception of one having a real input for the purpose of open loop analysis. With the boundary conditions from Table 1, the trajectories of the spool displacement and the pressures in the volumes are shown in Figure 6. Figure 6(a) shows an increase in the flow rate of 1% in the supply line pressure modeled as a step function. This increase in the supply flow increases the net pressure force in the spool valve and opens the valve more (Figure 6(b)) resulting in a larger flow area (Figure 6(c)) and hence more flow through the valve. This relieves the excess pressure in the supply line resulting in an increase in pressure of in the supply line of about 1.6% (Figure 6(d)).

### 3.1 Open Loop Stability Analysis of Pressure Relief Valve

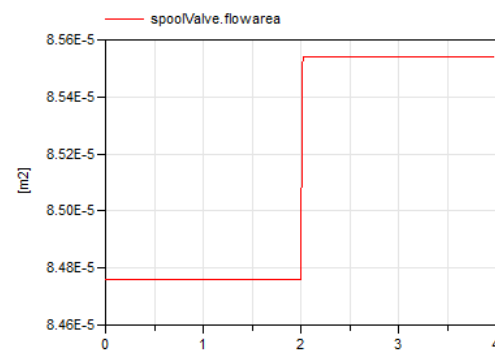
The dynamic model of the pressure relief valve in Figure 15 was perturbed from its steady state (Figure 6) by an increase in the supply pressure. This model was set-up at this steady state operating point and linearized for open loop analysis. The open loop model was set-up such that the pressure in the piston and the force the pressure exerts on the spool are decoupled (or “broken”). This means that the transients in the pressure are de-coupled from the pressure-force equation. This is an example of a feedback loop embedded within the physics of the model. There ap-



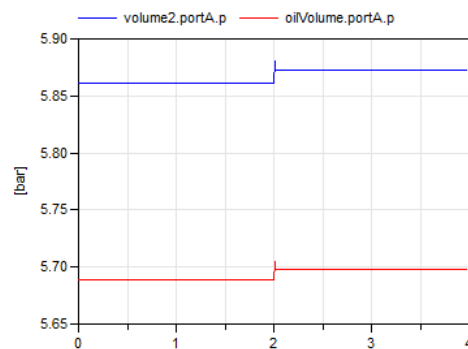
(a) Supply flow



(b) Spool displacement (also valve opening)



(c) Flow area of valve compared with maximum flow area of  $4.8919 \times 10^{-4} \text{m}^2$



(d) Supply line pressure

Figure 6: Trajectories of the inlet flow, line pressure and valve opening

appears to be no elegant way to break these loops for open loop controls analysis other than modifying the equations manually. The block diagram representation of the relief valve of Figure 5 is shown in Figure 7 [1]. The inner loop represents the change in the force due to change in the volume of the piston and the outer loop represents the change in the force due to the change in the supply line pressure. In this analysis, only the effect of the line pressure on the force needs to be decoupled and is "broken" by duplicating the section of the model connecting the supply line to the piston using a master-slave approach as labeled in Figure 17. In the master section, the line pressure is held at steady-state by blocking the transients whereas in the slave, the volume of the slave piston is held at steady-state while allowing the transients of the line pressure. The pressure-force in the slave piston is the output of the open loop system representing the breaking of the outer loop of the block diagram of Figure 7. The real input to this piston represents the input force represented as  $F_0$  in Figure 7. The linearized system has 6 states:  $x, \dot{x}, P_C, P_S, P_{val}, P_{CS}$  where the symbols correspond to those introduced in Table 1 with  $x, \dot{x}$  representing the spool displacement and velocity  $P_C, P_{CS}$  are the pressure in the master and slave piston connected to the damping orifice and  $P_{val}$  is the pressure of the fluid in the spool valve.

The state space system of the relief valve is represented as:

$$\begin{bmatrix} \dot{P}_C \\ \dot{P}_S \\ \dot{P}_{val} \\ \dot{x} \\ \dot{\dot{x}} \\ \dot{P}_{CS} \end{bmatrix} = A_{lin} \begin{bmatrix} P_C \\ P_S \\ P_{val} \\ x \\ \dot{x} \\ P_{CS} \end{bmatrix} + B_{lin} F_0$$

$$F_{out} = C_{lin} \begin{bmatrix} P_C \\ P_S \\ P_{val} \\ x \\ \dot{x} \\ P_{CS} \end{bmatrix} + D_{lin} F_0$$

where  $A_{lin} =$

$$\begin{bmatrix} -4.18 \times 10^3 & 0 & 0 & 0 & 7.17 \times 10^9 & 0 \\ 2.13 \times 10^4 & -7.74 \times 10^5 & 6.19 \times 10^5 & 0 & 0 & 0 \\ 0 & 7.74 \times 10^4 & -1.02 \times 10^5 & 1.94 \times 10^{13} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -0.068 & 0 & 0.0072 & -4.7857 \times 10^6 & 0 & 0 \\ 0 & 4.18 \times 10^3 & 0 & 0 & 0 & -4.18 \times 10^3 \end{bmatrix}$$

$$\text{and } B_{lin} = \begin{bmatrix} 0 & 0 & 0 & 0 & -199.998 & 0 \end{bmatrix}^T, C_{lin} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -3.42 \times 10^{-4} \end{bmatrix}.$$

The Bode plot of the above linearized system is shown in Figure 8.

Component	Parameter	Value
Spring Damper	Pre-load force $F_0$	150N
	Spring Constant $K_s$	10000N/m
Mass	m	0.005kg
Spool Valve	Chamber volume $V_c$	$3 \times 10^{-5} m^3$
	Spool area $A_s$	$3.4212 \times 10^{-4} m^2$
	Radial Clearance r	$1 \times 10^{-5} m$
Piston	Volume $V_d$	$5 \times 10^{-5} m^3$
	Area A	$3.4212 \times 10^{-4} m^2$
Supply flow	$Q_{in}$	$0.002 m^3 s^{-1}$
Load Pressure	$P_L$	5bar
Damping Orifice	Area $A_{do}$	$5 \times 10^{-7} m^2$
	Transition Pressure	0.1bar
Metering Orifice	Area $A_{mo}$	$5 \times 10^{-5} m^2$
Outlet Orifice	Area $A_{out}$	$2.5 \times 10^{-5} m^2$

Table 1: System parameters and boundary conditions

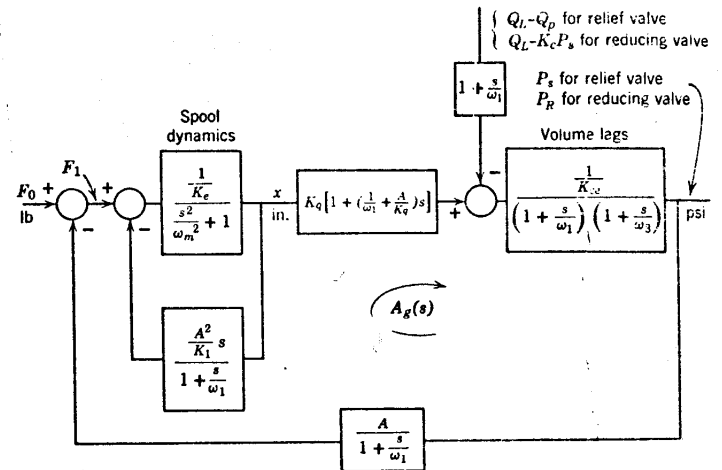


Figure 7: Block diagram representation of the relief valve

The steady state gain of the valve as seen from the magnitude plot of Figure 8 is affected by the flow-pressure coefficient of the spool valve and its volume at steady state. The coefficient is expressed as:

$$K_1 = \frac{q_{val,SS}}{2P_{val,SS}} \quad (2)$$

where  $q_{val,SS}$  and  $P_{val,SS}$  are the steady state flow rate and pressure through the spool valve respectively. This coefficient is directly affected by the geometric parameters of the valve and the type of valve opening (see Section 2.3) which gives the relationship between the spool displacement and the flow area. Similarly, the sizing of the orifices in the relief valve model also affects the corresponding flow-pressure coefficients which then impact the frequency response and

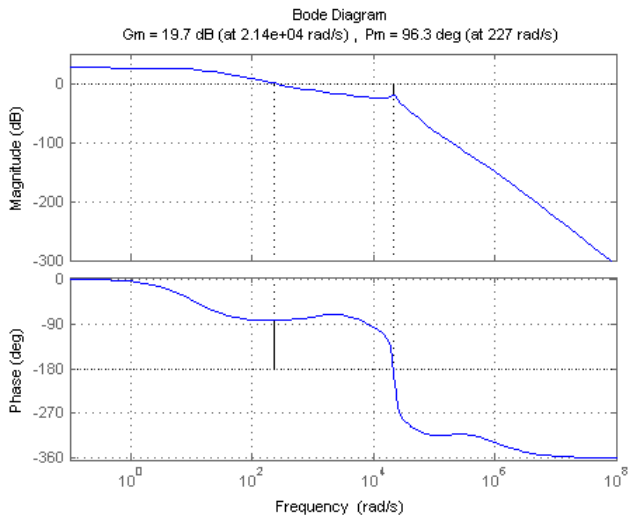


Figure 8: Bode plot of the linearized pressure relief valve system

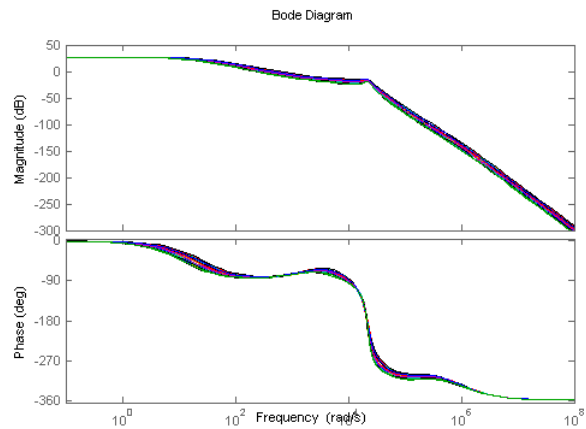
therefore the stability characteristics. Further design rules for relief valves can be found in [1].

### 3.2 Robust design study of the relief valve model

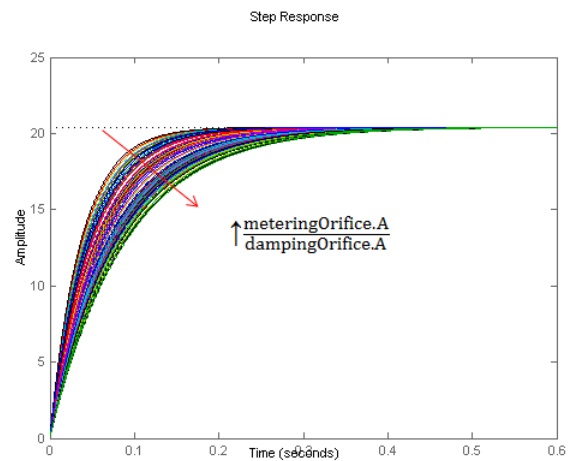
The effect of design parameters on the gain and the phase margin of the system is discussed next. A Functional Mock-up Unit (FMU) of the model is imported into MATLAB using the FMI Toolbox for MATLAB. FMIT has tools for conducting Design of Experiments [5] and a simple design of experiment is illustrated here to show the effect of the geometric parameters of the valve on the stability characteristics.

The orifice areas of the damping and outlet orifices in the PRV model of Figure 17 are set with respect to a constant factor of the area of the metering orifice. These multiplicative factors are then varied in the DoE and the changes in the frequency response analyzed. First, the damping orifice area is varied through 100 points uniformly distributed between factors of 0.01 to 0.025 in comparison to the metering orifice area which is constant at  $5 \times 10^{-5} \text{m}^2$ .

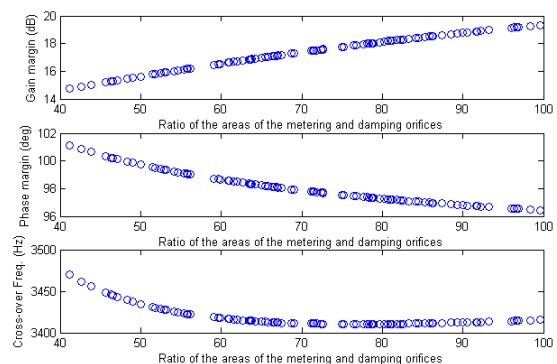
Figure 9 shows the bode plots, step response and the variance in the gain, phase margins and the cross-over frequency for different sizes of the damping orifice. The bode plot of Figure 9(a) shows differences in the open-loop gain at higher frequencies. While the damping orifice area has no influence on the steady-state gain, it can be observed that the response time is clearly affected by the damping orifice. As the damping orifice area increases, the damping ratio of the system also increases. The gain margin and phase



(a) Bode plot



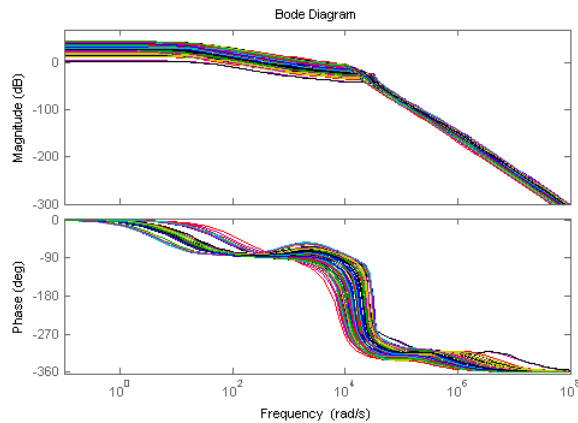
(b) Step response



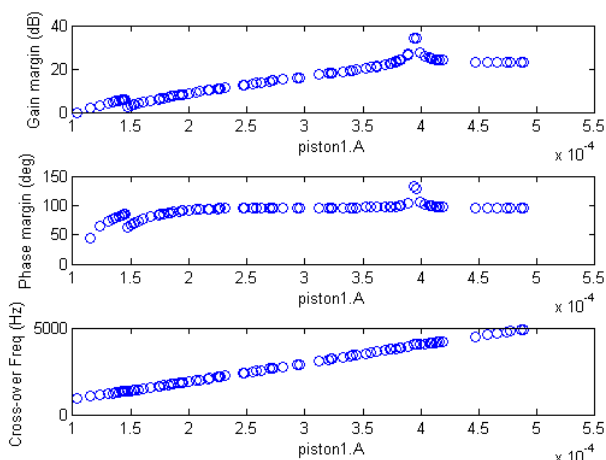
(c) Gain, phase margins and cross-over frequency

Figure 9: Effect of the damping orifice area on the frequency response





(a) Bode plot



(b) Gain, phase margins and cross-over frequency

Figure 10: Effect of the piston area on the frequency response

margins also do not show significant sensitivities to changes in the damping orifice area as seen by a change of 6dB and 6 deg in these quantities respectively.

The next design parameter of interest is the area of the piston in the valve. It is assumed that the pistons in the spool valve, and both pistons in the PRV model of Figure 15 are assumed to have equal areas and for the sake of simplicity, the rod areas in these pistons are assumed to be negligible. The piston areas are then varied over 100 uniformly distributed points across a range of  $1 \times 10^{-4}$  to  $5 \times 10^{-4} \text{m}^2$  and linearized about the respective steady state points.

The bode plot of Figure 10(a) shows that there is a significant sensitivity in the steady state gain to changes in the piston area. In fact, the steady state gain decreases with increasing piston area with iden-

tical damping characteristics. The gain and the phase margin (Figure 10(b)) also show significant increase with increasing piston areas. A suitable choice of piston area can be made from this plot with a gain of 3dB (or more) and a phase margin between 30 and 60 degrees to result in reasonable trade-offs between bandwidth and stability.

## 4 Modeling and Open Loop Stability Analysis of an Electro-Hydraulic Servo Valve

A single stage Electro-Hydraulic servo valves is modeled in this work with a torque motor directly attached to a 2-land 4-way directional control valve. A representation of this valve used to actuate an aircraft control surface is shown in Figure 11 [2].

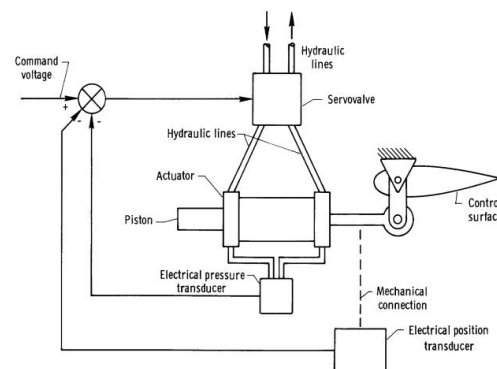


Figure 11: Schematic of a single stage servo valve to actuate an aircraft control surface

The directional control valve used in this hydraulic actuation system was modeled using a combination of volumes, spool valves and pistons from the Elements package and is shown in Figure 12.

Each pair of the spool valve and the piston are identical and the sizing of these are identical to that of the pressure relief valve discussed in Table 1. The complete model of the system is shown in Figure 16.

The function of the torque motor in Figure 16 is to convert the input voltage into the corresponding spool displacement to displace the directional control valve by an appropriate amount in the desired direction to open one of the spool valves that connect port P to port A or port P to port B for flow from port P which correspondingly opens the spool valves that connect port B to port T and port A to port T for flow into port T. Ports A and B are connected to the two hydraulic ports of a

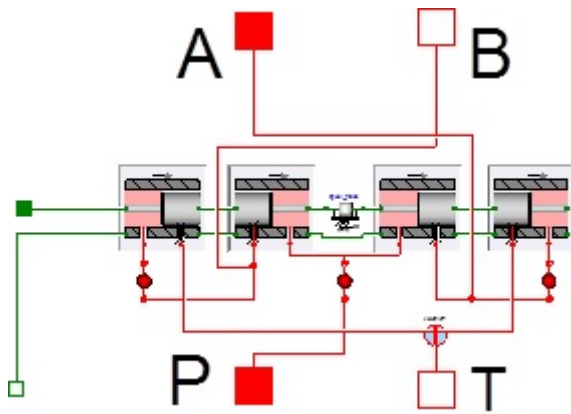


Figure 12: Model of a direction control valve made from components of the Elements package

differential cylinder to convert the fluid power into mechanical displacement to actuate the control surface.

The torque motor dynamics are not modeled in this work for simplicity but are represented as the following transfer function [2]

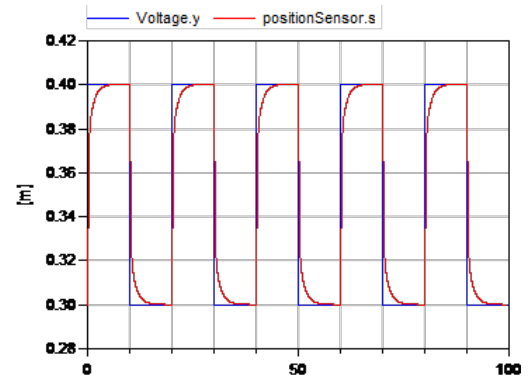
$$X_s(s) = \frac{0.003}{0.0025s + 1} E(s) \quad (3)$$

where  $X_s(s)$  and  $E(s)$  are the Laplace transform representation of the spool displacement of the directional control valve and the voltage signal to the torque motor respectively.

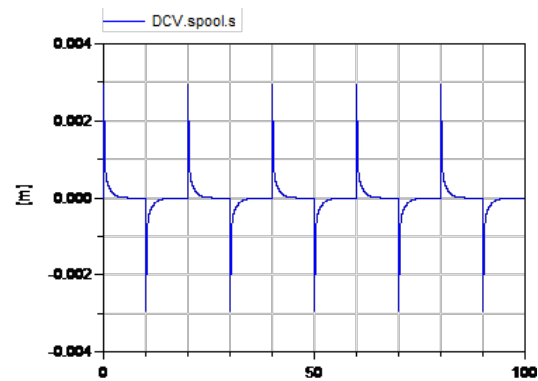
The pressure differential in the cylinder moves the piston against the preload force of the spring. This displacement of the piston in the cylinder is then sensed and acts as the feedback signal. The error between the input signal and the piston position is then amplified to be the voltage input of the torque motor.

The piston is assumed to be at a steady state at a position of 0.3 m. A pulse signal is sent as the input and the trajectories of the spool displacement of the directional control valve and that of the piston are plotted in Figure 13.

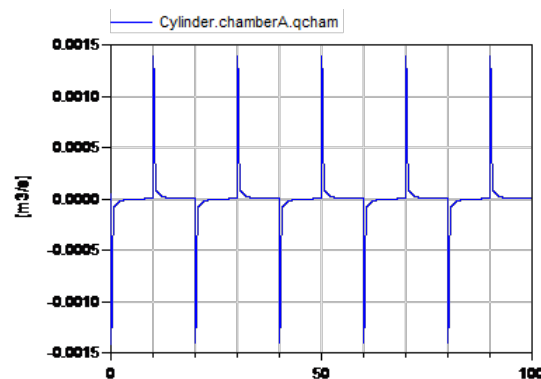
From Figure 13(a), as the piston is commanded to shift in the positive direction, the torque motor causes the directional control valve spool to displace in the positive direction (Figure 13(b)) to open the spool valves connecting ports P to A and ports B to T. The flow from port P through port A enters the piston and increases the pressure in that chamber to cause an increase in the net force on the spring to push the piston in the commanded direction. The pulse command shows the movement of the piston in both directions and the flows into the cylinder from the valve are plotted in Figure 13(c).



(a) Piston position and command



(b) Spool displacement of DCV



(c) Flow rate into cylinder

Figure 13: Trajectories of the piston, spool positions and flow rate into cylinder

The stability characteristics can be studied by breaking the feedback loop and setting the piston position as the output and the commanded position as the input. The model is linearized and the Bode plot of Figure 14 shows the frequency response of the system.

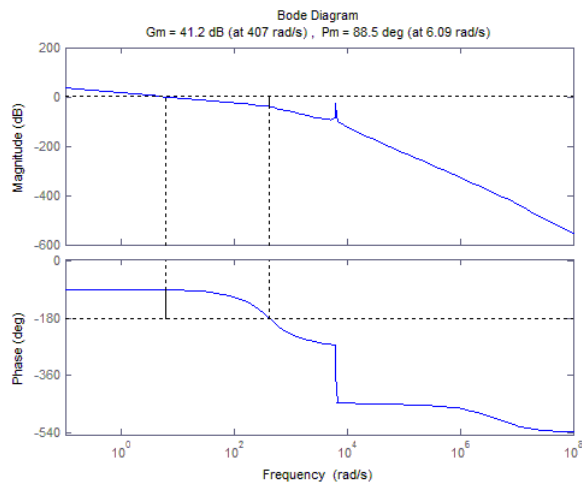


Figure 14: Bode plot of the linearized servo valve actuation system

It might be noted that the loop opening in this example is straightforward since the control law is outside of the physics of the model. In the case of the relief valve, the pressure force equation is part of the physics of the valve and there appears to be no elegant way with any Modelica based tool to break these loops other than to manually modifying the equations to enable the model to be used for controls analysis. For modeling, Modelica provides the great advantage that the modeler does not have to transform the model manually to block diagram form. For open loop control analysis, and a feedback loop that is embedded into the physics, the modeler will have to derive a block diagram form also in Modelica in order to obtain the open loop equations. It is however possible to integrate the open loop variant for certain typical feedback situations in library models. This will be made available in the upcoming version of the Hydraulics Library. In a block diagram based modeling environment like Simulink where there is no clear distinction between the physics of the model and the control system, such loop breakage is straight forward. The Simulink feature to automatically apply the steady-state value as offset at the input at which the loop is broken should also be available in Modelica tools.

It may however be possible to provide improved automated support for open loop stability analysis in Modelica tools, because the symbolic processing in

Modelica compilers for efficient code generation is very similar to the causalization process when deriving a block diagram from model equations. Another issue with open loop analysis of hydraulic components is that SI units, which are typically used in the model equations, are very badly scaled for the open loop analysis. The scaling that Modelica tools apply internally through the use of the nominal attribute does not help, because for the stability analysis the user would like to have properly scaled equations that are consistent between the linearized system and the closed loop model.

## 5 Conclusion

In this paper, we discussed the physical design of hydraulic components using the Hydraulics library and illustrated the application of robust design techniques to improve the physical design of valves with Modelica and FMI based tools. The application of the Elements package demonstrated both the modeling of relief valves and also in constructing more complex components from the basic building blocks already available from the library. We also highlighted a limitation in controls analysis of models where the feedback loops are embedded within the physics. There is a new paradigm developing in emerging physical systems where the feedback loops are embedded tightly within the physics of these systems and it is imperative that these aspects are addressed in Modelica and its associated tools.

## References

- [1] Merritt, H.E., "Hydraulic Control Systems", *John Wiley Sons, Inc.*, 1967.
- [2] Edwards J.W., "Analysis of an Electro-Hydraulic Aircraft Control-Surface Servo and Comparison with Test Results", *NASA Technical Note NASA TN D-6928*, 1972.
- [3] Eaton Corp., "Product Literature Library Proportional Directional Valves without Feedback, (V-VLPO-MC007-E), Proportional Valves with Feedback, (V-VLPO-MC005-E)", *Accessed from [http://hydraulics.eaton.com/products/valves\\_proportional\\_directional\\_controls.htm](http://hydraulics.eaton.com/products/valves_proportional_directional_controls.htm)*, in Nov 2013.
- [4] Tchkalov V., Miller S., "Parameterization of Directional and Proportional Valves in Sim-Hydraulics", *Accessed from*

<http://www.mathworks.com/matlabcentral/fileexchange/27260> in Nov 2013.

- [5] Modelon AB, "FMI Toolbox for MATLAB", Accessed from <http://www.modelon.com/products/fmi-toolbox-for-matlab/> in Nov 2013.
- [6] Henningson M., Akesson J., Tummescheit H., "FMI-Based Tools for Robust Design of Dynamical Systems", *Proceedings of the 10th International Modelica Conference*, Lund 2014.

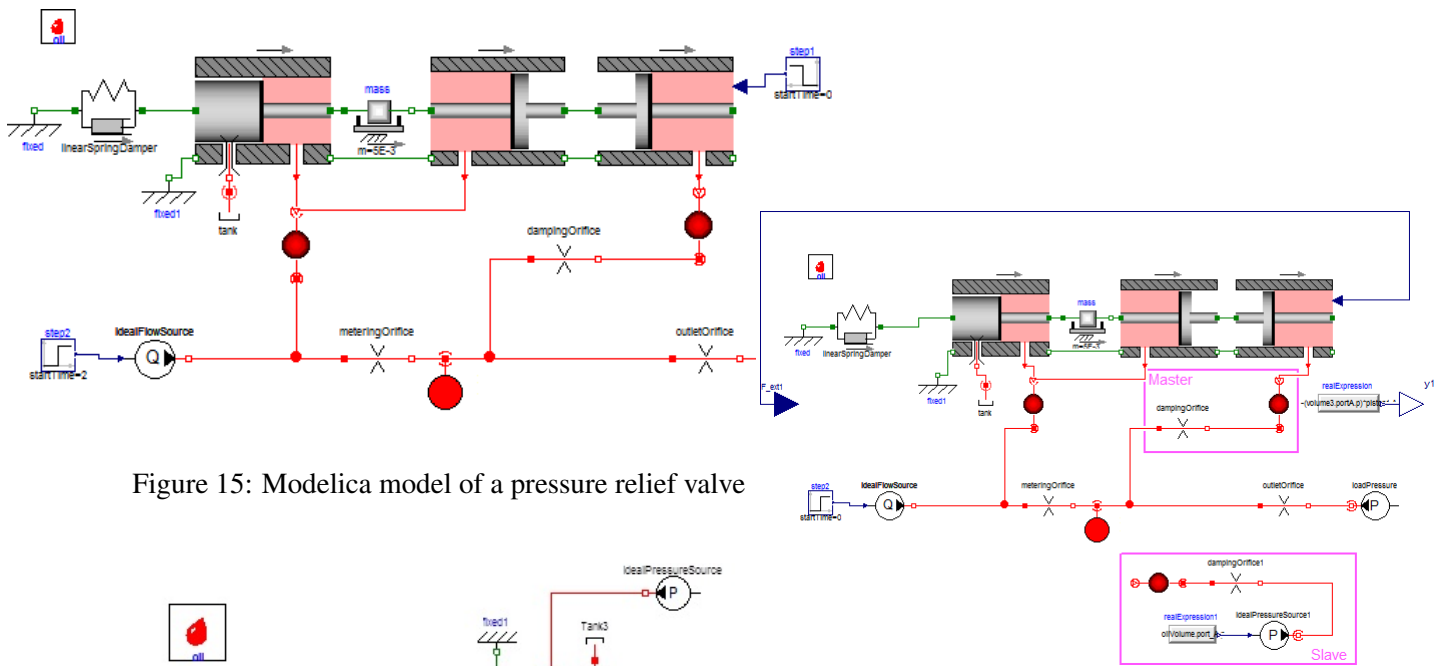


Figure 15: Modelica model of a pressure relief valve

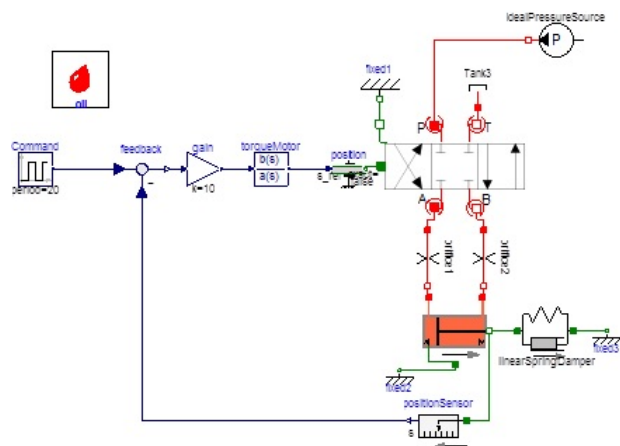


Figure 16: Modelica model of an Electro-hydraulic servo valve

Figure 17: Open loop model of the PRV system

# Exploiting Actuator Limits with Feedforward Control based on Inverse Models

Manuel Gräber  
TLK-Thermo GmbH  
Hans-Sommer-Str. 5, 38106 Braunschweig, Germany

## Abstract

Feedforward control based on inverse dynamic plant models (linear or nonlinear) is a suitable method to enhance set-point tracking performance of control systems. In reality actuators always have limits, but limiting functions can not be inverted. A common approach to handle this issue is to invert the unlimited plant model and detune the feedforward filter in order to stay always in between the actuator limits. This approach causes a loss in performance for rapid set-point changes, because the actuator range is not entirely used. In this article a rather simple but powerful method is presented, which overcomes this performance issue for many types of plant models. Actuator limits are fully exploited, and the obtained trajectories are close to optimal ones. Simulation and measurement results demonstrate the usability of the proposed feedforward structure.

*Keywords: set-point tracking; model inversion; actuator saturation; nonlinear control; model-based control; anti-windup*

## 1 Introduction

Assuming the dynamic behaviour of a plant is known, and a control task is to drive the plant output to a given set-point or track a given trajectory. A reasonable idea is to use the known plant behaviour to compute the needed plant input. Mathematically spoken a dynamic plant model is inverted. The original physical model computes the model output as reaction to the plant input. Whereas the inverse model computes the plant input for a given output. Modelica as equation-based modeling language provides a powerful possibility to automatically generate inverse dynamic models. In many cases inverse models can directly be derived from forward models. Details about model inversion with Modelica can be found in [11].

There are different ideas, where to place the inverse

model inside a control loop. In this paper we concentrate on the most obvious variant: using the inverse model in the feedforward path.

Feedforward control based on inverse dynamic plant models (linear or nonlinear) has been proven to be a suitable method to enhance set-point tracking performance of control systems. Successful applications have been reported in [8, 5].

But there is one issue, that often dramatically decreases the performance of feedforward designs: actuator limits. Every real-world control system has limited actuators. That means the control signal  $u(t)$  is constrained by

$$u_{\min} \leq u(t) \leq u_{\max} \quad (1)$$

The problem with Equation (1) is, that it is not invertible. A common approach is, to not include input constraints and invert an unconstrained plant model. But what happens if the unconstrained inverse model in a feedforward path computes plant inputs which violate constraints (1)? A straightforward solution is sketched in Figure 1. A filter  $F$  and the inverse plant model  $\tilde{G}^{-1}$  is used to compute an input signal out of reference signal  $r$ . A limiter finally cuts off the unconstrained input signal  $v$  at its limits and feeds the input signal  $u$  to the plant  $G$ .

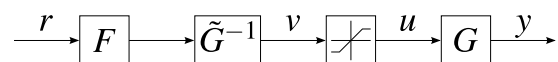


Figure 1: Feedforward control with inverse plant model and actuator limitation.

It's easy to imagine, that if we apply a step jump to  $r$  this can lead to very large values of  $v$  at the first time points. Its magnitude of course depends on the inverse model and the filter. Theoretically, for most plant models a step jump of  $r$  with no filter ( $F = 1$ ) would even lead to an infinite impulse of  $v$ . But large values of  $v$  generate *actuator saturation*, that means  $u$  is equal  $u_{\max}$  or  $u_{\min}$ . The resulting difference between

$u$  and  $v$  causes a inconsistency between the states of the inverse plant model and the plant. Which directly means performance decrease of the feedforward control. This effect can be interpreted as *windup*, which is well known for integral controllers. In Figure 4 example simulation results are plotted, where saturation of  $u$  leads to unwanted oscillation of plant output  $y$ .

Well-known anti-windup techniques for feedback controllers, which can be described in state-space form, are conditioning [7] or the more general observer-based approach described in [1]. Although these schemes are developed for feedback controllers they could also be used for feedforward control. In fact the proposed method in this article shows similarities to observer-based approaches, but as described in Section 2 there is one important difference.

Straightforward approaches to avoid windup for inverse models as feedforward are: filter detuning or application dependent trajectory planning (e.g. set-point ramps). Drawbacks of these methods are the lack of generality and a loss of performance – at least if simple approaches are used. In [11] handling actuator limits is briefly discussed, although it is not the main scope of this article. As one possibility it is suggested to provide feasible reference trajectories by online or offline solving of dynamic optimization problems. This approach provides high performance potential. Due to its optimal character actuator limits can be fully exploited. But dynamic optimization is usually a non-trivial task.

In [10] actuator saturation is addressed with feedback from the plant according to *internal model control*. This general approach has the advantage to be not application specific. And with only one remaining tuning parameter its simple to parameterize. As a result the output of the inverse model  $v$  stays within its limits but does not fully exploit them.

Instead of detuning the filter  $F$  or using application dependent trajectory planning, a more general method to avoid windup is presented in this article. The basic concepts and ideas are described in Section 2. No theoretical proof of stability is given, but numerical experiments in Section 4 and real-world experiments in Section 5 demonstrate the usability of the proposed feedforward scheme. Benefits of this approach are:

- applicable to nonlinear (SISO) systems
- easy to set up using Modelica
- only one tuning parameter: filter frequency
- full exploitation of actuator limits, almost time-optimal trajectories

## 2 Feedforward with Actuator Limits

The basic idea of the new control structure is sketched in Figure 2. The original feedforward system in Figure 1 is complemented with an internal feedback loop. Similar to anti-windup schemes for integral controllers [1] the actuator saturation signal  $v - u$  is fed back with a constant gain  $k$  to the input signal.

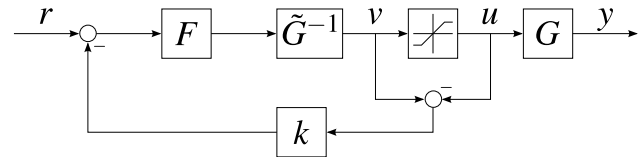


Figure 2: Proposed feedforward control with internal anti-windup feedback.

If the unconstrained signal  $v$  is beyond its limits the negative feedback will decrease the windup effect and the constraint violation. The actual performance of this compensation strongly depends on the filter  $F$  and the gain factor  $k$ . In the following we will design  $F$  and  $k$  in a way, that there will practically no constraint violation of  $v$ .

The basic idea is to use a very high gain  $k$  to drive constraints violations to zero. See [12] for details on high gain feedback for nonlinear systems. In order to achieve stability of the feedforward loop with high gains, the open loop  $F\tilde{G}^{-1}$  has to fulfill a few requirements:

1.  $F\tilde{G}^{-1}$  must be stable.
2.  $F\tilde{G}^{-1}$  must have stable zero dynamics (minimum phase for linear systems).
3. Relative degree of  $F\tilde{G}^{-1}$  must be equal zero.

Condition 3 actually is stricter than needed. For stability of the high gain feedback loop a relative degree equal one would be sufficient. But numerical experiments show, that the feedforward performance is much better, if we restrict the relative degree to zero. Resulting in a direct algebraic feed through from  $F$ 's input to  $\tilde{G}^{-1}$ 's output and a nonlinear implicit algebraic equation of the overall feedforward system of equations (see Section 3). This is a crucial feature of the proposed implicit method, which distinguishes this method from observer-based explicit anti-windup schemes described in [1].

The open loop requirements can be transformed into requirements for  $F$  and  $\tilde{G}$ . Condition 1 and 2 hold also for the classical feedforward control of Figure 1 and

lead to the requirements that plant model  $G$  must be stable and must have stable zero dynamics. Condition 3 can easily be fulfilled if we choose the order of the filter  $F$  equal to the relative degree of  $\tilde{G}$ .

1.  $\tilde{G}$  must have stable zero dynamics.
2.  $\tilde{G}$  must be stable.
3. Order of Filter  $F$  must be equal to the relative degree of  $\tilde{G}$ .

It is important to note, that these requirements are not (yet) theoretical proven sufficient conditions. They are motivated by the theory of high-gain feedback and successful numerical experiments with various linear and nonlinear plant models.

### 3 Feedforward control using Modelica

Modelica provides a convenient possibility to generate feedforward control laws based on the proposed method. Due to its equation-based design the error-prone plant model inversion is handled by the Modelica compiler. Using the free Modelica Standard Library (MSL) and a given forward plant model, this task can even be done graphically. In Figure 3 the graphical representation of an example feedforward control system is shown. The plant is modelled as linear second order system, but it could be any other SISO Modelica model. This structure corresponds exactly to Figure 2. Plant model inversion is easily defined by exchanging input and output with the MSL block `InverseBlockConstraints`. See [11] for further details on model inversion with Modelica.

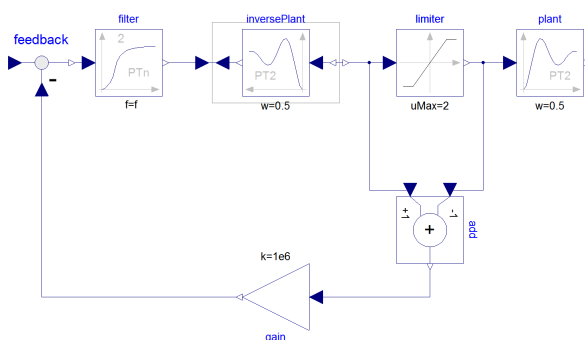


Figure 3: Graphical Modelica implementation of the proposed feedforward structure.

According to the requirements in Section 2 the chosen filter order is two – identical to the plant model's

relative degree. This means there is a direct algebraic relation between the filter input and the output of the inverse plant model. Using the notation of Figure 2 this relation can be expressed as function  $f$  with

$$v = f(r - k(v - u), \mathbf{x}) \quad (2)$$

where  $\mathbf{x}$  is the vector of internal states of  $F\tilde{G}^{-1}$ . Together with Equation (1) we obtain a nonlinear system of equations

$$v = f(r - k(v - u), \mathbf{x}) \quad (3a)$$

$$u = \begin{cases} u_{\max} & v > u_{\max} \\ u_{\min} & v < u_{\min} \\ v & \text{else} \end{cases} \quad (3b)$$

which can be reduced to one unknown  $v$ . To conclude this analysis: the resulting anti-windup feedforward control law is a system of differential and algebraic equations (DAE), with at least one nonlinear algebraic equation.

A straight-forward way to test this control law at a real plant is to use Functional Mock-up Unit (FMU) [3] export capabilities of Modelica tools. The resulting FMU for model-exchange comes with an embedded solver for implicit equations. The drawback is, that one has no control of the internal iterative solution process. But at least for prototyping purposes this approach works very well. In Section 5 a FMU exported from Dymola is used as controller at a real plant. With TLK-Thermo's FMI Suite the FMU is imported into the data acquisition and control environment LabView.

## 4 Simulation Experiments

In this section we look at two different linear example plants, modelled with the Modelica Standard Library. We study the effect of different filter parameters, which is the remaining degree of freedom of the proposed feedforward control. Finally optimal control trajectories are presented and compared to the obtained ones.

### 4.1 Plant Models and Simulation Setup

The chosen example plant models are two stable but differently damped second order systems. Angular frequency is 0.5 and gain is 1.0 in both cases. The resulting transfer functions are of the form

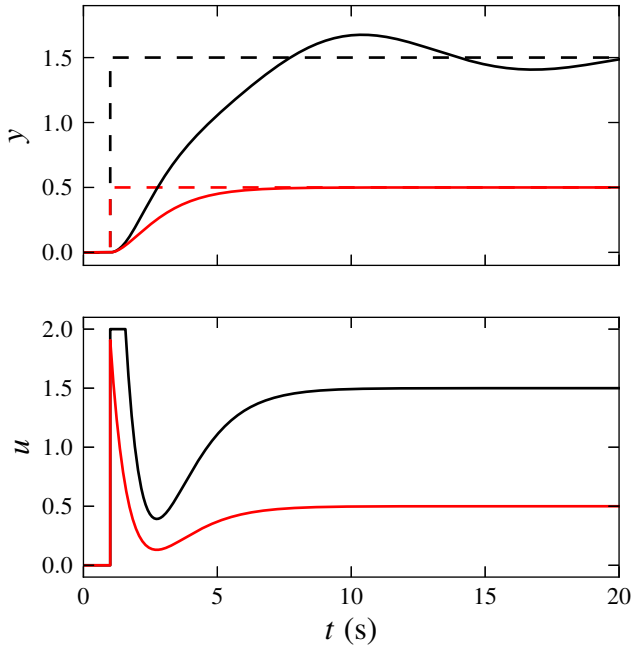


Figure 4: Plant input and output response to reference steps of different height. Conventional feedforward design.

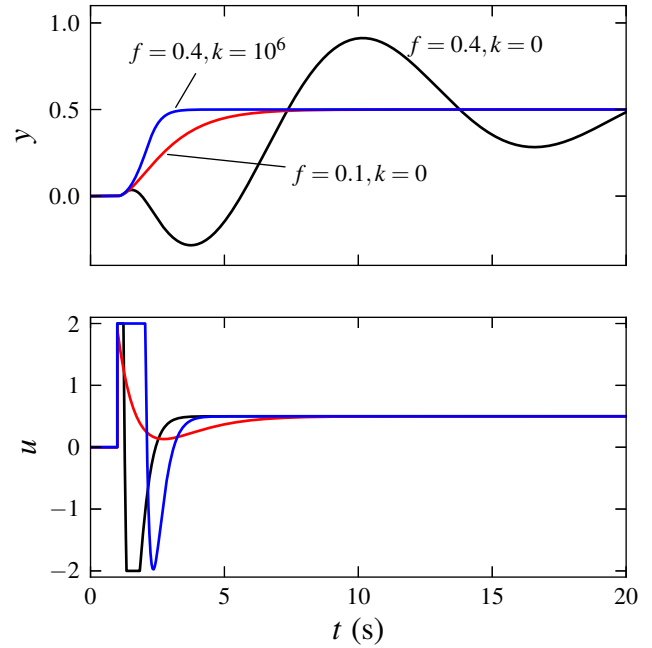


Figure 5: Plant input and output response to a reference step. Conventional feedforward ( $k = 0$ ) with different filter cut-off frequencies  $f$  (Hz) compared to proposed anti-windup design ( $k = 10^6$ ).

$$G(s) = \frac{1}{4s^2 + 4Ds + 1}$$

with

$$D = 0.2 \quad (4a)$$

and

$$D = 1.2 \quad (4b)$$

Plant models and the proposed feedforward control are implemented in Modelica using MSL blocks. The graphical representation of the total simulation model is shown in Figure 3. For both plants actuator limits are  $-2$  and  $2$ . Gain  $k$  of the internal feedback loop is set to a comparable high number of  $10^6$ , which ensures that  $v$  virtually stays in between the actuator limits. According to the requirements in Section 2 the filter order is two.

## 4.2 Filter Tuning

The remaining degree of freedom for designing of the feedforward control is the filter's cut-off frequency  $f$ . In general higher values for  $f$  lead to a more aggressive feedforward control. This behaviour is identical

for the proposed anti-windup scheme as well as for the common scheme of Figure 1. In the latter case one would choose the filter frequency, so that the unconstrained plant input  $v$  would stay inside the actuator limits. There are two drawbacks with this approach. First, by not exploiting actuator limits we lose performance in terms of rise or settling time. Second, given a filter frequency  $f_1$  works well for a reference step of 1, it is by no means guaranteed, that it does also work for higher step jumps. Instead of using a pure linear filter one would need to implement an application dependent algorithm for *Trajectory Planning*.

The proposed additional internal feedback (see Figure 2) solves both issues. Actuator limits are fully exploited using a general not application specific approach. In the following this is illustrated with comparative simulation results for both feedforward schemes with the slightly damped plant model (4a).

In Figure 4 plant input and output responses to reference steps using the conventional feedforward scheme ( $k = 0$ ) are plotted. In both cases the filter frequency is 0.1 Hz. For a reference step of 0.5 (red) the input signal  $u$  stays inside the actuator limits. Whereas a reference step of 1.5 (black) leads temporarily to unconstrained input  $v$  above  $u_{\max}$ . The plant input  $u$  is chopped off at  $u_{\max}$ , which leads to a mismatch be-



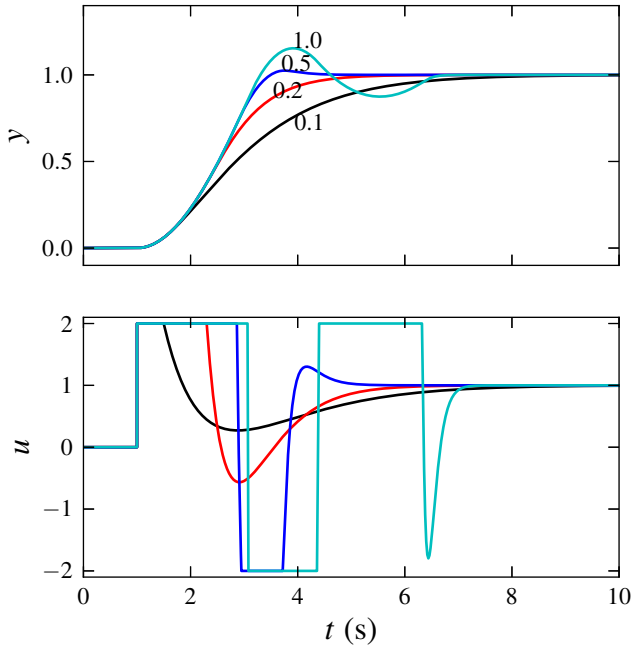


Figure 6: Response to reference unit step at  $t = 1$  for different filter cut-off frequencies (in Hz). Slightly damped plant model with  $D = 0.2$ .

tween the states of the inverse plant model and the plant. This wind-up effect can be observed at the plant output  $y$  in form of unwanted oscillations (black).

Using internal high-gain feedback wind-up can be avoided. As a result we can use higher filter frequencies and fully exploit actuator limits. In Figure 5 simulation results for a step height of 0.5 are shown. The red curves are identical to Figure 4. If we increase the filter frequency to 0.4 Hz (black) we observe actuator saturation and oscillations of  $y$ . Things change dramatically if we turn on feedback with  $k = 10^6$  (blue). Now, plant input  $u$  also hits its limit, but plant output  $y$  behaves perfectly smooth. Compared to feedforward without feedback (red) the settling time is decreased by 63%.

A natural question is, what happens if the filter frequency is further increased? An answer is given in Figure 6. Using identical plant and feedforward setup as in the simulations above, the filter frequency is changed. Increasing  $f$  from 0.1 to 0.2 Hz leads to a better feedforward performance. At 0.5 Hz we already see a small overshoot of  $y$ , and at 1.0 Hz there are notable oscillations in  $u$  and  $y$ .

Simulation experiments with the well-damped plant model (4b) give similar results, plotted in Figure 7. Here, we can choose higher filter frequencies. At  $f = 1$  Hz there is still no overshoot of  $y$ . Further increase

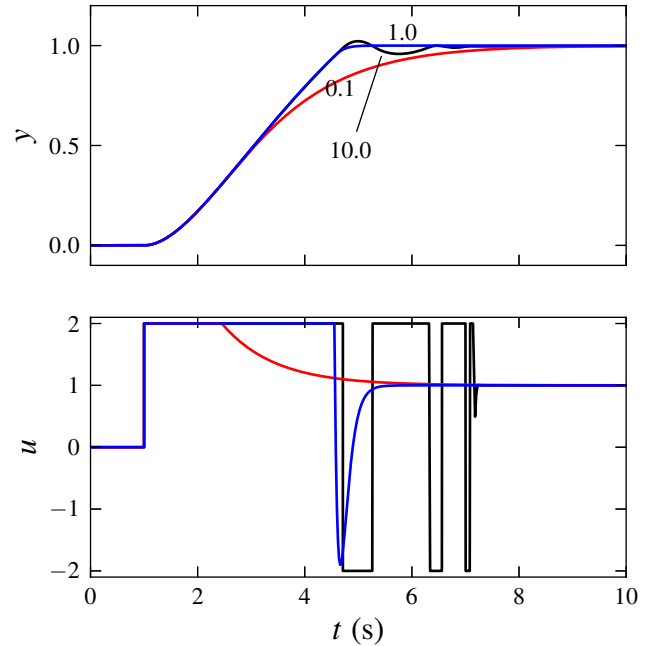


Figure 7: Response to reference unit step at  $t = 1$  for different filter cut-off frequencies (in Hz). Well damped plant model with  $D = 1.2$ .

of  $f$  finally also leads to oscillations in  $u$  and  $y$ .

To conclude these observations: Depending on the plant dynamics there seems to be a critical filter frequency  $f_{\text{crit}}$ . Values of  $f$  greater than this critical frequency lead to overshooting or even oscillating plant outputs.

As conclusion from additional simulation experiments, that are not shown in this article:  $f_{\text{crit}}$  strongly depends on the plant dynamics. Whereas dependencies to actuator limits and the reference signal (e.g. step height) seem to be very weak.

As consequence of these observations we formulate a tuning rule for filter frequency  $f$ :

- Simulate a step jump of the reference signal with the given plant model.
- Increase  $f$  until overshooting of the plant output is observed. This is  $f_{\text{crit}}$ .
- Choose  $f$  to be less than  $f_{\text{crit}}$ . Smaller values lead to increased robustness against model uncertainty.

This tuning rule is by no means theoretically founded. It is purely derived from simulation experiments.

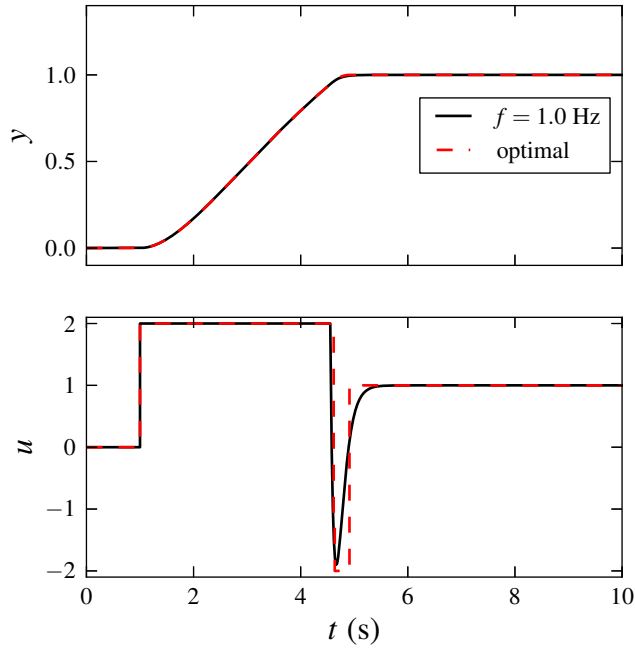


Figure 8: Time-optimal trajectories of plant input and output for rapid set-point changes compared to trajectories obtained with anti-windup feedforward. Well-damped plant model.

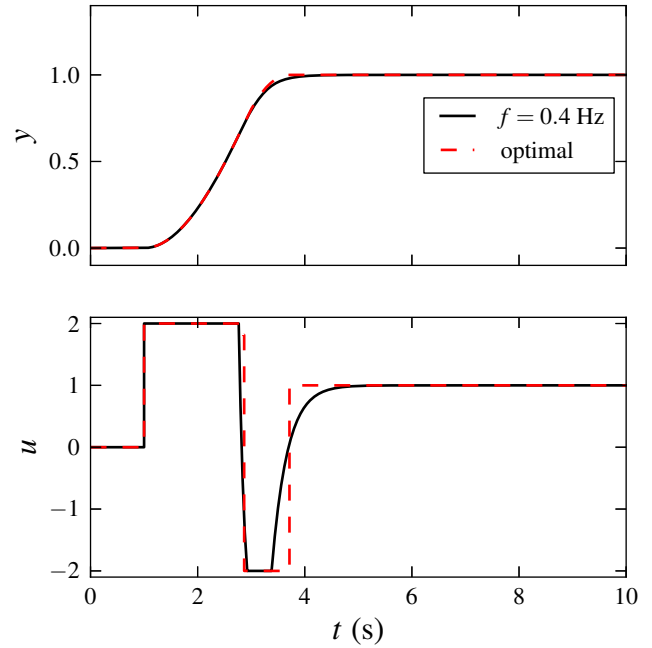


Figure 9: Time-optimal trajectories of plant input and output for rapid set-point changes compared to trajectories obtained with anti-windup feedforward. Slightly-damped plant model.

### 4.3 Comparison with Optimal Trajectories

In Section 4.2 several control trajectories are obtained with the proposed feedforward algorithm. Now we want to find out, how good they are. More precisely, we compare them with optimal trajectories, defined by the solution of the *Optimal Control Problem*

$$\min_{\mathbf{x}(\cdot), u(\cdot)} t_e \quad (5a)$$

$$\text{s.t.} \quad \frac{d\mathbf{x}}{dt}(t) = \mathbf{f}(\mathbf{x}(t), u(t)) \quad (5b)$$

$$-2 \leq u(t) \leq 2 \quad (5c)$$

$$\mathbf{x}(0) = \mathbf{0} \quad (5d)$$

$$\frac{d\mathbf{x}}{dt}(t_e) = \mathbf{0} \quad (5e)$$

$$y(t_e) = 1 \quad (5f)$$

$\mathbf{x}$  denotes the vector of state variables and  $\mathbf{f}$  is the corresponding ODE function of plant model (4). Optimization task is to drive the plant from  $\mathbf{x}(0) = \mathbf{0}$  as fast as possible to steady state with  $y(t_e) = 1$ .

OCP (5) is solved numerically using *Direct Multiple Shooting* within the software package MUSCOD-II [4, 6, 9]. The obtained optimal trajectories of  $u$  and

$y$  are plotted for both plants in Figure 8 and 9 together with feedforward trajectories. Frequencies of feedforward filter are chosen according to simulation results from Section 4.2. One can see that there are differences of the feedforward control inputs  $u$  to its optimal trajectories. But at the output  $y$  of the well-damped plant Figure 8 there is practically no deviation between optimal and feedforward values. In Figure 9 deviations at the output of the slightly-damped plant can be detected, but they are comparable small. For both plant models the proposed feedforward control provides input trajectories for an almost time-optimal set-point change.

## 5 Real-World Experiments

In this section the proposed feedforward algorithm is tested on a real plant. To eliminate set-point tracking errors from model/plant mismatch feedforward control is enhanced with additional feedback. Measurement results are presented for pure feedforward and combined feedback/feedforward control.

## 5.1 Electric Water Heater

The experimental plant is a water heater as part of an existing test stand, where it is used to provide water flow at a given temperature. As sketched in Figure 10 a water mass flow  $\dot{m}$  with inlet temperature  $T_{in}$  is heated by electrical power  $P_{el}$  and leaves the heater with outlet temperature  $T_{out}$ . The electrical heating power is thyristor controlled with an analog input signal  $u$ . Control task is set-point tracking of plant output  $y = T_{out}$  using input  $u$ .  $\dot{m}$  and  $T_{in}$  can be seen as disturbances.

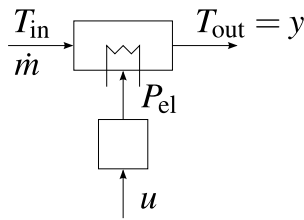


Figure 10: Schematic view of the controlled plant.

## 5.2 Plant Model

In Modelica a mixed physical/empirical model of the electric water heater is developed. The static behavior is modelled with the energy balance

$$0 = \dot{m}c_p(T_{in} - T_{out}) + P_{el} \quad (6)$$

and a nonlinear empirical relation

$$P_{el} = f(x) \quad (7)$$

where  $x$  is an internal state. Equation (7) describes the nonlinear static behavior of the phase cutting thyristor controller.

The overall plant dynamics are modelled as second order time delay model. The identified transfer function from  $u$  to  $x$  is

$$G_{ux}(s) = \frac{1}{(1 + 5.804s)(1 + 5.809s)} e^{-1.885s} \quad (8)$$

In Figure 11 the measured step response used for identification together with the model step response is shown. Good agreement between fitted model and measured dynamics can be noticed.

## 5.3 Feedforward Measurement Results

Measurement results of the proposed feedforward control applied to the real plant are plotted in Figure 13.

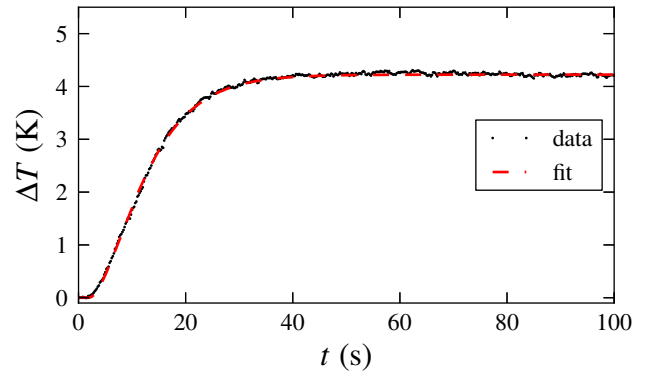


Figure 11: Measured input step response compared to the response of the identified second order time delay model.

The shown experiment spans set-point steps of varying height. Similar to the simulation experiments the feedforward reacts very aggressively to set-point changes and exploits actuator limits (0-10V). Due to model plant mismatch the output does not exactly track the given set-points. The static error is 1-2 K. Compared to the full control range of 27 K this error is relatively small.

If one would still like to drive this static error to zero, the feedforward control can be combined with measurement feedback, as it is done in the next section.

## 5.4 Combined Feedforward and Feedback Control

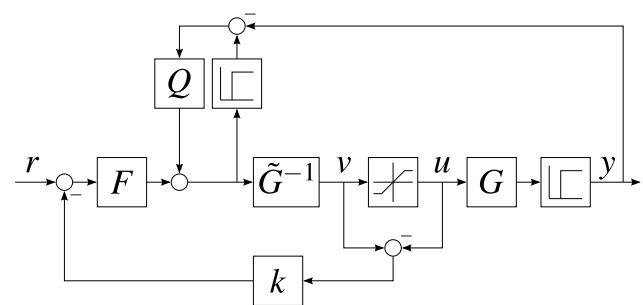


Figure 12: Feedforward and feedback control structure.

There are different possibilities, how to combine feedforward and feedback in one control system. See for example [11] for a brief overview. Following the approach suggested in [2] we add the output of an feedback filter  $Q$  to the input of the inverse plant model. The input of  $Q$  is the difference between measured and desired plant output. The resulting control

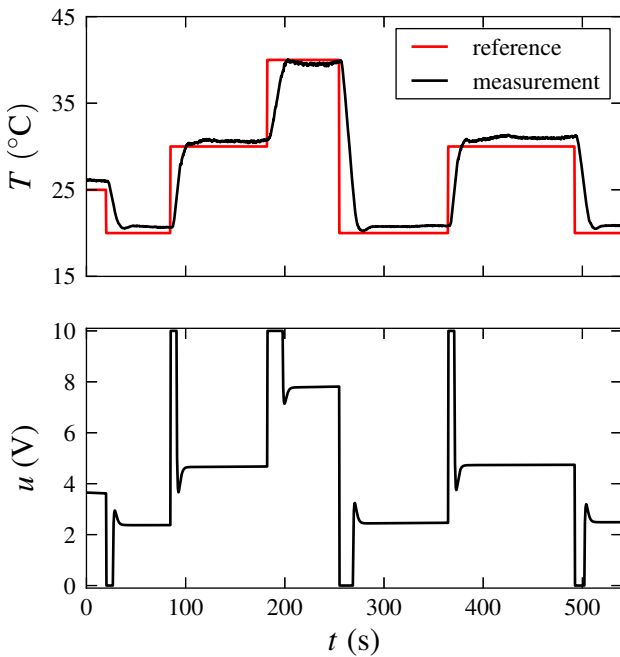


Figure 13: Measurement results from feedforward control of a liquid heater.

structure is sketched in Figure 12. For  $Q = 1$  infinity-gain feedback is obtained. Usually  $Q$  is implemented as low pass filter with unity gain. Filter order and frequency are tuning parameters of the feedback loop. Compared to [2] the control structure in Figure 12 has two additional elements. One is the internal feedback loop of the feedforward path as it is discussed in the previous sections. The second addition is a delay block. It is necessary because, plant model (8) includes delay, which is not invertible. This issue can be solved by splitting the model in a continuous block without delay and an additional delay block. Only the continuous block is inverted. The delay block is added afterwards to the desired plant output, before it is compared with the measured real output.

Measurement results obtained with this control structure are shown in Figure 14. As in Figure 13 the closed loop shows fast reaction to set-point changes. Additionally the static control error is eliminated.

## 6 Conclusion

The proposed anti-windup scheme for feedforward control with inverse models leads to near optimal control trajectories for a broad class of plant models: stable nonlinear SISO systems with stable zero dynamics. Because of its equation-based nature Modelica provides an user-friendly possibility to generate ap-

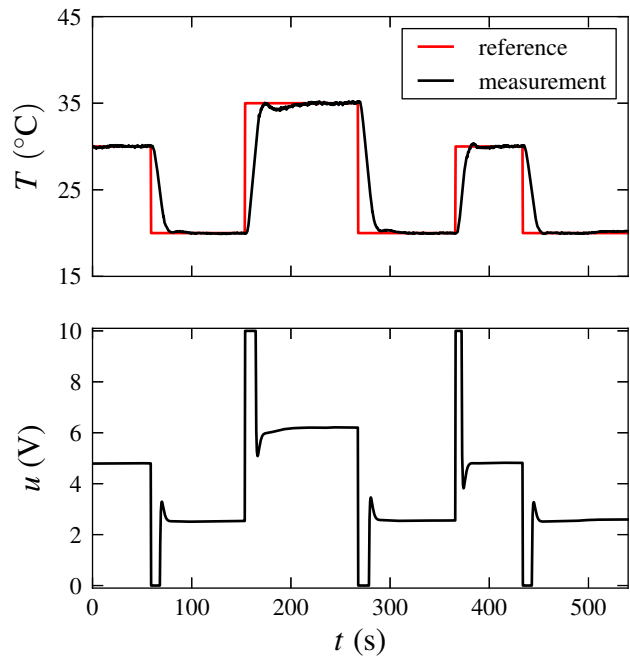


Figure 14: Measurement results from combined feedforward and feedback control of a liquid heater.

propriate feedforward control laws based on (forward) system models of the controlled plant. With one remaining tuning parameter – filter cut-off frequency – the proposed method is easy to configure. Experiments with a real plant show good performance of the anti-windup feedforward control. Control errors due to model plant mismatch are successfully avoided by extending feedforward control with feedback loop.

## Acknowledgments

This work is part of the research project "Reflex Thermo" funded by the German Federal Ministry of Education and Research (reference EM1STR002).

## References

- [1] Karl Johan Åström and Lars Rundqwist. Integrator windup and how to avoid it. In *Proc. of the American Control Conference*, 1989.
- [2] N. Bajcinca and T. Bünte. A novel control structure for dynamic inversion and tracking tasks. In *Proc. of the 16th IFAC World Congress*, Prague, 2005.
- [3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss,

- M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proc of the 8th International Modelica Conference*, Dresden, 2011.
- [4] H. G. Bock and K. J. Plitt. A Multiple Shooting algorithm for direct solution of optimal control problems. In *Proc. of the 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.
- [5] G.M. Clayton, S. Tien, a.J. Fleming, S.O.R. Moheimani, and S. Devasia. Inverse-feedforward of charge-controlled piezopositioners. *Mechatronics*, 18(5-6):273–281, June 2008.
- [6] Moritz Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*. Phd thesis, Universität Heidelberg, 2001.
- [7] R. Hanus, M. Kinnaert, and J.-L. Henrotte. Conditioning technique, a general anti-windup and bumpless transfer method. *Automatica*, 23(6):729–739, 1987.
- [8] Stephen W John, Gursel Alici, and Christopher D Cook. Inversion-based feedforward control of polypyrrole trilayer bender actuators. *IEEE/ASME Transactions on Mechatronics*, 15(1):149–156, 2010.
- [9] D B Leineweber, I Bauer, A A S Schäfer, H G Bock, and J P Schlöder. An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization (Parts I and II). *Computers and Chemical Engineering*, 27:157–174, 2003.
- [10] Chung Seng Ling, Michael D Brown, Paul F Weston, and Clive Roberts. Gain Tuned Internal Model Control for Handling Saturation in Actuators. In *Proc. of the American Control Conference*, Boston, 2004.
- [11] Gertjan Looye, Michael Thümmel, Matthias Kurze, Martin Otter, and Johann Bals. Nonlinear Inverse Models for Control. In *Proc. of the 4th International Modelica Conference*, Hamburg, 2005.
- [12] Riccardo Marino. High-gain feedback in nonlinear control systems. *International Journal of Control*, 42(6):1369–1385, 1985.



# An FMI-based Framework for State and Parameter Estimation

Marco Bonvini, Michael Wetter, Michael D. Sohn  
Simulation Research Group, Lawrence Berkeley National Laboratory  
1 Cyclotron Road, 94720, Berkeley, CA

## Abstract

This paper proposes a solution for creating a model-based state and parameter estimator for dynamic systems described using the FMI standard. This work uses a nonlinear state estimation technique called unscented Kalman filter (UKF), together with a smoother that improves the reliability of the estimation. The algorithm can be used to support advanced control techniques (e.g., adaptive control) or for fault detection and diagnostics (FDD). This work extends the capabilities of any modeling framework compliant with the FMI standard version 1.0.

*Keywords: Nonlinear State and Parameter Estimation; Unscented Kalman Filter (UKF); Smoothing; Functional Mockup Interface (FMI); Fault Detection and Diagnosis (FDD)*

## 1 Introduction

In many applications, after the system has been designed, controls and/or fault detection and diagnostics (FDD) algorithms are developed and deployed. These techniques should be able to leverage the models developed during the earlier design stages, thereby increasing the productivity of the overall product development. Advanced control (such as adaptive control or model predictive control) and FDD techniques require an enhanced knowledge of the system state. For example, the flight controller of an airplane should try to estimate the real velocity and position of the aircraft while compensating for measurements errors and sensor noises. When dealing with dynamic system, having an enhanced knowledge about the system state means estimating its state variables with associated error bounds.

This paper proposes a solution for creating a model-based state estimator for dynamic systems described using the FMI standard. This work extends the capabilities of any modeling framework compliant with the FMI standard version 1.0. The FMI is a stan-

dard that allows to embed a simulation model within a unified interface in order to couple simulation models developed using different simulation programs. Although the FMI standard has been created mainly for co-simulation, we leverage this standard for providing an algorithm that is compatible with a large number of simulation and modeling platforms, including Modelica-based ones.

There are several characteristics intrinsic of any model-based state estimation technique. These characteristics are related to the model properties (e.g., the Kalman filter is applicable just to linear models), the assumptions introduced when describing the probability distribution of the state variables (e.g., assuming they are Gaussian) and the computational performance of the underlying algorithm (e.g., the number of simulations or computations to be done in order to provide an estimation).

The state estimation technique used in this work is the unscented Kalman filter (UKF) [1, 2]. The UKF is able to deal with nonlinear systems and it just requires to perform function evaluations of the model in order to compute the evolution of its state variables and the value of its outputs. The UKF has less requirements about the knowledge of the model with respect to other nonlinear state estimation techniques. For example the extended Kalman filter needs to linearize the model [3]. The computational performances of the UKF are modest with respect to other Monte Carlo based techniques (like particle filters [4]), enabling its use for real-time applications.

The proposed work leverages the UKF technique and provides a state and parameter estimation algorithm for a dynamic system (e.g., modeled with Modelica or Matlab) embedded according to the FMI standard as a Functional Mockup Unit (FMU). The model, once exported as FMU can be used to set up a state and parameter estimator to

- calibrate the model during the commissioning phase in order to check if it performs as expected,
- compute a probabilistic estimation of unknown

variables of the system (e.g., observable but not measured) for control or FDD.

The paper is structured as follows. Section 2 starts with a brief introduction about the state estimation and it continues with a description of the unscented Kalman filter and smoothing algorithm, including the modifications required to extend the state estimation procedure to the parameters. Subsection 2.5 gives more information about the implementation details, while subsection 2.6 contains a code snippet that shows how to use the proposed algorithm. Section 3 contains an example that shows how the proposed algorithm can be used to identify faulty operation in a valve in the presence of measurement errors.

## 2 Method

The FMI-based state and parameter estimation algorithm consists of two components: (i) a filter for the state and parameter estimation, (ii) a smoother that improves the quality of the estimation when the measurements are noisy and sometimes erroneous. The proposed algorithm has been written in Python and uses PyFMI [5]. Some of the basic methods and classes provided by PyFMI have been extended to fit our purposes. For example, we modified how FMUs are executed in parallel.

### 2.1 State Estimation

Kalman Filter (KF) [3] are often used to estimate state variables. However, as they are only applicable for linear systems, they are not suited for our applications. For systems that are described by nonlinear differential equations, the state estimation problem can be solved using an Extended Kalman Filter (EKF) [3]. The EKF linearizes around the current state estimate the original nonlinear model. However, in some cases, this linearization introduces large errors in the estimated second order statistics of the estimated state vector probability distribution [6]. Another approach is to simulate sample paths that generate random points in the neighborhood of the old posterior probability, for example by using Monte Carlo sampling, and adopting particle filters for the state estimation [4]. These techniques are robust with respect to model nonlinearities, but they are computationally expensive. The UKF faces the problem representing the state as a Gaussian random variable, the distribution of which is modeled non-parametrically using a set of points known as sigma points [1]. Using the sigma points, i.e., by propagating

a suitable number of state realizations through the state and output equations, the mean and the covariance of the state can be captured. The favorable properties of the UKF makes its computational cost far lower than the Monte Carlo approaches, since a limited and deterministic number of samples are required. Furthermore, the UKF requirements fit perfectly with the infrastructure provided by PyFMI since it provides an interface to the FMU model that allows to set state variables, parameter and running simulations.

### 2.2 The Unscented Kalman Filter

The Unscented Kalman Filter is a model based-techniques that recursively estimates the states (and with some modifications also parameters) of a nonlinear, dynamic, discrete-time system. This system may for example represent a building, an HVAC plant or a chiller. The state and output equations are

$$\mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t), t) + \mathbf{q}(t_k), \quad (1a)$$

$$\mathbf{y}(t_k) = H(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t), t) + \mathbf{r}(t_k), \quad (1b)$$

with initial conditions  $\mathbf{x}(t_0) = \mathbf{x}_0$ , where  $f: \mathfrak{R}^n \times \mathfrak{R}^m \times \mathfrak{R}^p \times \mathfrak{R} \rightarrow \mathfrak{R}^n$  is nonlinear,  $\mathbf{x}(\cdot) \in \mathbb{R}^n$  is the state vector,  $\mathbf{u}(\cdot) \in \mathbb{R}^m$  is the input vector,  $\Theta(\cdot) \in \mathbb{R}^p$  is the parameter vector,  $\mathbf{q}(\cdot) \in \mathbb{R}^n$  represents the process noise (i.e. unmodeled dynamics and other uncertainties),  $\mathbf{y}(\cdot) \in \mathbb{R}^o$  is the output vector,  $H: \mathfrak{R}^n \times \mathfrak{R}^m \times \mathfrak{R}^p \times \mathfrak{R} \rightarrow \mathfrak{R}^o$  is the output measurement function and  $\mathbf{r}(\cdot) \in \mathbb{R}^o$  is the measurement noise.

The UKF is based on the typical prediction-correction style methods:

1. PREDICTION STEP: predict the state and output at time step  $t_{k+1}$  by using the parameters and states at  $t_k$ .
2. CORRECTION STEP: given the measurements at time  $t_{k+1}$ , update the posterior probability, or uncertainty, of the states prediction using Bayes' rule.

The original formulation of the UKF imposes some restrictions on the model because the system needs to be described by a system of initial-value, explicit difference equations (1). A second drawback is that the explicit discrete time system in (1) cannot be used to simulate stiff systems efficiently. The UKF should be translated in a form that is able to deal with continuous time models, possibly including events.

Although physical systems are often described using continuous time models, sensors routinely report



time-sampled values of the measured quantity (e.g. temperatures, pressures, positions, velocities, etc.). These sampled signals represent the available information about the system operation and they are used by the UKF to compute an estimation for the state variables.

A more natural formulation of the problem is represented by the following continuous-discrete time model

$$\frac{d\mathbf{x}(t)}{dt} = F(\mathbf{x}(t), \mathbf{u}(t), \Theta(t), t), \quad (2a)$$

$$\mathbf{x}(t_0) = \mathbf{x}_0, \quad (2b)$$

$$\mathbf{y}(t_k) = H(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t), t_k) + \mathbf{r}(t_k), \quad (2c)$$

where the model is defined in the continuous time domain, but the outputs are considered as discrete time signals sampled at discrete time instants  $t_k$ . The original problem described in equation (1) can be easily derived as

$$\begin{aligned} \mathbf{x}(t_{k+1}) &= f(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t_k), t_k) \\ &= \mathbf{x}(t_k) + \\ &\quad \int_{t_k}^{t_{k+1}} F(\mathbf{x}(t), \mathbf{u}(t), \Theta(t), t) dt \end{aligned} \quad (3)$$

Our implementation uses this continuous-discrete time formulation and the numerical integration is done using PyFMI that works with a model embedded as an FMU. Despite not shown in (2) and (3) the model may contain events that are handled by the numerical solver provided with the PyFMI package.

The UKF is based on the the Unscented Transformation [1] (UT), which uses a fixed (and typically low) number of deterministically chosen sigma-points<sup>1</sup> to express the mean and covariance of the original distribution of the state variables  $\mathbf{x}(\cdot)$ , exactly, under the assumption that the uncertainties and noise are Gaussian [1]. These sigma-points are then propagated simulating the nonlinear model (2) and the mean and covariance of the state variables are estimated from them. This is significantly different from Monte Carlo approaches because the UKF chooses the points in a deterministic way. One of the most important properties of this approach is that if the prior estimation is distributed as a Gaussian random variable, the sigma points are the minimum amount of information needed to compute the exact mean and covariance of the posterior after the propagation through the nonlinear state function [6].

<sup>1</sup>The sigma-points can be seen as the counterpart of the particles used in Monte Carlo methods.

Figure 1 illustrates the filtering process which we will now explain. At time  $t_k$ , a measurement of the outputs  $\mathbf{y}(t_k)$ , the inputs and the previous estimation of the state are available. Simulations are performed starting from the prior knowledge of the state  $\hat{\mathbf{x}}(t_{k-1})$ , using the input  $\mathbf{u}(t_{k-1})$ . Once the results of the simulations  $\hat{x}_{sim}(t_k)$  and  $\hat{y}_{sim}(t_k)$  are available, they are compared against the available measurements in order to correct the state estimation. The corrected value (i.e. filtered) becomes the actual estimation. Because of its speed, the estimation can provide near-real-time updates, since the time spent for simulating the system and correcting the estimation is typically shorter than the sampling time step, in particular for building or HVAC applications, where computations take fractions of second and sampling intervals are seconds or minutes.

The Algorithm 1 summarize the steps performed by the UKF. The interested reader can find more information and details of the actual implementation in [7].

### 2.3 Smoothing to Improve UKF Estimation

In this subsection, we discuss an additional refinement procedure to the UKF. The distribution  $P(\mathbf{x}(t_k) | \mathbf{y}(t_1), \dots, \mathbf{y}(t_k))$  is the probability to observe the state vector  $\mathbf{x}(t_k)$  at time  $t_k$  given all the measurements collected. By using more data  $P(\mathbf{x}(t_k) | \mathbf{y}(t_1), \dots, \mathbf{y}(t_k), \dots, \mathbf{y}(t_{k+N}))$ , the posterior distribution can be improved through recursive smoothing. Hence, the basic idea behind the recursive smoothing process is to incorporate more measurements before providing an estimation of the state. Figure 2 represents the smoothing process. While the filter works forwardly on the data available, and recursively provides a state estimation, a smoothing procedure back-propagates the information obtained during the filtering process, after some amount of data becomes available, in order to improved the estimation previously provided [8].

The smoothing process can be viewed as a delayed, but improved, estimation of the state variables. The longer the acceptable delay, the bigger the improvement since more information can be used. For example if, at a given time, a sensor provides a wrong measurement, the filter may not be aware of this and it may provide an estimation that does not correspond to the real value (although the uncertainty bounds will still be correct). The smoother observes the trend of the estimation will reduce this impact of the erroneous data, thus providing an estimation that is less sensitive to measurement errors.

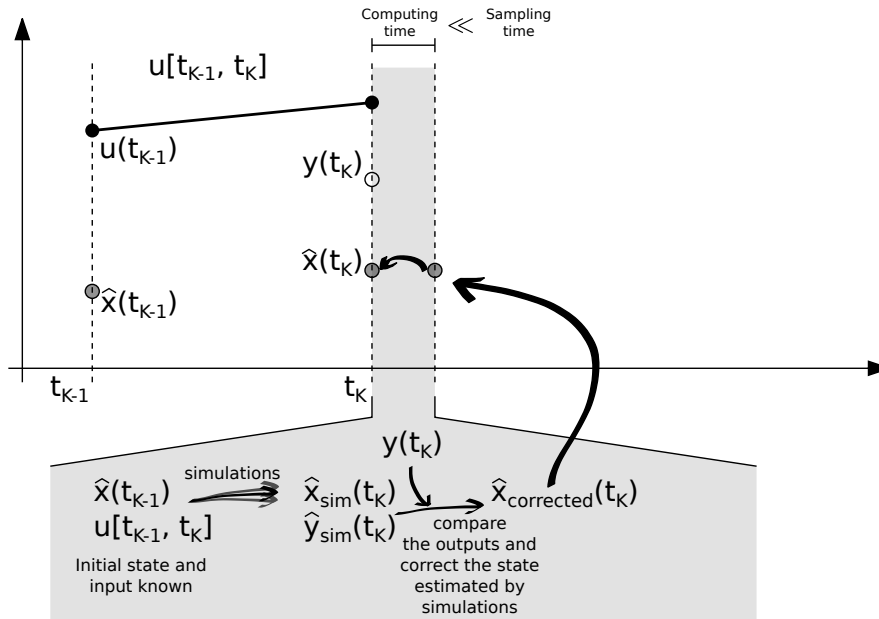


Figure 1: On-line state estimation filtering procedure.

Procedurally, the smoothing algorithm starts from a user-specified point in the data stream and back updates the previously filtered estimation. Algorithm 2 describes the smoothing procedure.

### 2.4 Parameter Estimation

The importance of the state estimation has been stressed, and we described the UKF and Smoother as solutions to this problem. While state estimation is particularly important for controls, parameter estimation is important for model calibration and fault detection and diagnostics. Consider, for example an heat exchanger. Suppose it is characterized by one heat exchange coefficient that influences the heat transfer rate between the two fluids. During the design of the heat exchanger it is possible to compute an approximation of it. However, it is not possible to know its value exactly. After the heat exchanger is created, identifying the value of it is important to verify if the design requirements have been met. Another example is real-time monitoring in which it is continuously monitored during the operation in order to continuously check if it has been reduced by fouling and the heat exchanger need to be serviced.

Continuous parameter estimation is possible by extending the capabilities of the UKF and Smoother to estimate not just the state variable, but also the parameters of the system. The approach is to include the parameter in an augmented state  $\mathbf{x}^A(\cdot)$ , defined as

$$\mathbf{x}^A(\cdot) = [\mathbf{x}(\cdot) \quad \mathbf{x}^P(\cdot)]^T, \quad (4)$$

where  $\mathbf{x}^P(\cdot) \subseteq \Theta(\cdot)$  is a vector containing a subset of the full parameter vector  $\Theta(\cdot)$  to be estimated. The new components of the state variables need a function that describe their dynamics. Since in the normal operation, these values are constant, the associated dynamic is

$$\frac{d \mathbf{x}^P(t)}{dt} = \mathbf{0}, \quad (5)$$

where  $\mathbf{0}$  is a null vector. These null dynamics have to be added (2). The result is a new continuous-discrete time system

$$\frac{d \mathbf{x}^A(t)}{dt} = F_A(\mathbf{x}^A(t), \mathbf{u}(t), \Theta(t), t) \quad (6a)$$

$$\mathbf{y}(t_k) = H(\mathbf{x}^A(t_k), \mathbf{u}(t_k), \Theta(t_k), t_k) + \mathbf{r}(t_k), \quad (6b)$$

with

$$F_A(\mathbf{x}^A(t), \mathbf{u}(t), \Theta(t), t) = \begin{bmatrix} F(\mathbf{x}(t), \mathbf{u}(t), \Theta(t), t) \\ \mathbf{0} \end{bmatrix} \quad (7)$$

Note that augmenting the state variables leads to a nonlinear state equation even if  $F(\cdot, \cdot, \cdot, \cdot)$  is a linear function. Therefore, for parameter estimation, a nonlinear filtering and smoothing technique is required.

### 2.5 Implementation

The former sections explained on the state and parameter estimation. This section describes the software implementation and describes specific issues that we addressed.

**Algorithm 1: Filtering**

Notation: The superscript  $(i)$  indicates that the quantity is related to the  $i$ -th sigma point,  $w_m^{(i)}$  and  $w_c^{(i)}$  are the weights associated to the  $i$ -th sigma point,  $n$  is the dimension of the state vector  $\mathbf{x}(\cdot)$ , and  $[\cdot]_i$  is an operator that if applied to a matrix  $A$  returns its  $i$ -th column. Vectors are indicated with bold characters.

Given the initial knowledge of the state distribution  $\mathbf{x}(t_0) \sim N(\mu_0, P_0)$ , and given the output measurement covariance matrix  $R_0$

1. Initialize  $k = 0$  and set parameters  $\alpha, \beta, \lambda$  (with  $0 \leq \alpha \leq 1$  – other configuration details in [2])
2. Define  $2n + 1$  sigma-points

$$\begin{aligned} \mathbf{x}(t_k)^{(0)} &= \boldsymbol{\mu}_k, \\ \mathbf{x}(t_k)^{(i)} &= \boldsymbol{\mu}_k + \left[ \sqrt{(n+\lambda)P_k} \right]_i, \quad i = 1 \dots n, \\ \mathbf{x}(t_k)^{(i)} &= \boldsymbol{\mu}_k - \left[ \sqrt{(n+\lambda)P_k} \right]_{i-n}, \quad i = n+1 \dots 2n. \end{aligned}$$

3. Compute the weights associated to each sigma-point

$$\begin{aligned} w_m^{(0)} &= \lambda / (n + \lambda), \\ w_c^{(0)} &= \lambda / (n + \lambda) + (1 - \alpha^2 + \beta), \\ w_m^{(i)} &= 1 / 2(n + \lambda), \quad i = 1 \dots 2n, \\ w_c^{(i)} &= 1 / 2(n + \lambda), \quad i = 1 \dots 2n. \end{aligned}$$

4. Compute the predicted state (i.e. perform a simulation) for each sigma-point, and the predicted weighted mean, and the predicted covariance

$$\begin{aligned} \mathbf{x}(t_{k+1})^{(i)} &= f(\mathbf{x}(t_k)^{(i)}, \mathbf{u}(t_k), \Theta(t_k), t_k), \quad i = 0 \dots 2n+1, \\ \boldsymbol{\mu}_{k+1}^- &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{x}(t_{k+1})^{(i)}, \\ P_{k+1}^- &= P_k + \sum_{i=0}^{2n+1} w_c^{(i)} \left( \mathbf{x}(t_{k+1})^{(i)} - \boldsymbol{\mu}_{k+1}^- \right) \left( \mathbf{x}(t_{k+1})^{(i)} - \boldsymbol{\mu}_{k+1}^- \right)^T. \end{aligned}$$

5. Redefine the new sigma-points  $\mathbf{x}(t_{k+1})^{(i)}$  using the predicted mean  $\boldsymbol{\mu}_{k+1}^-$ , and covariance  $P_{k+1}^-$  as shown in step (1).
6. Compute the measured outputs using the new sigma-points, then compute the mean output  $\hat{\mathbf{y}}_{k+1}$  and its covariance  $S_{k+1}$

$$\begin{aligned} \mathbf{y}(t_{k+1})^{(i)} &= H(\mathbf{x}(t_{k+1})^{(i)}, \mathbf{u}(t_{k+1}), \Theta(t_{k+1}), t_{k+1}), \quad i = 0 \dots 2n+1, \\ \hat{\mathbf{y}}_{k+1} &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{y}(t_{k+1})^{(i)}, \\ S_{k+1} &= R_0 + \sum_{i=0}^{2n+1} w_c^{(i)} \left( \mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1} \right) \left( \mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1} \right)^T. \end{aligned}$$

7. Compute the cross covariance between the state and the output

$$C_{k+1} = \sum_{i=0}^{2n+1} w_c^{(i)} \left( \mathbf{x}(t_{k+1})^{(i)} - \boldsymbol{\mu}_{k+1}^- \right) \left( \mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1} \right)^T.$$

8. Compute the filter gain and update the predicted mean and covariance of the state

$$\begin{aligned} K &= C_{k+1} S_{k+1}^{-1}, \\ \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_{k+1}^- + K [\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1}], \\ P_{k+1} &= P_{k+1}^- - K S_{k+1} K^T. \end{aligned}$$

9. Compute the state estimation as  $\hat{\mathbf{x}}(t_{k+1}) \sim N(\boldsymbol{\mu}_{k+1}, P_{k+1})$ .

10. Increment  $k$ , and go to step (2).

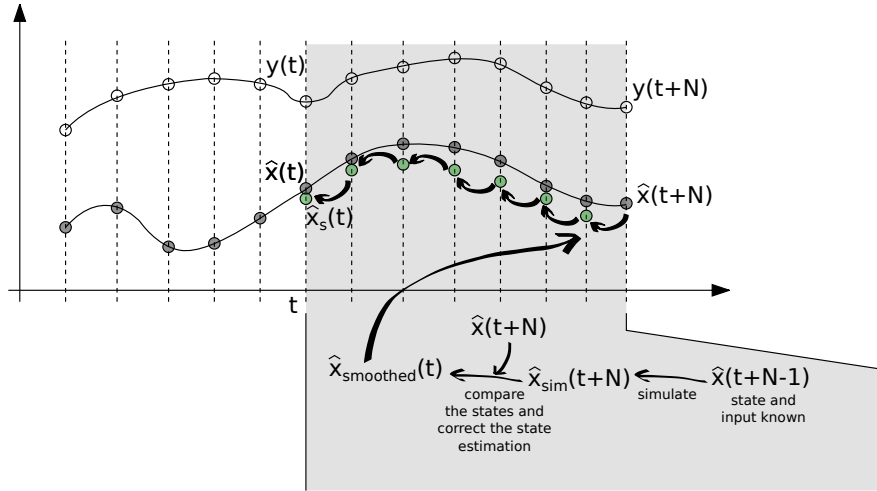


Figure 2: Improving the estimation using the smoothing. Backward propagations of the filtering results can improve the former estimation.

Algorithm 2: Smoothing

1. Initialize  $k \geq 0$  and define the amplitude of the smoothing window  $N > 0$  such as data at time  $t_{k+N}$  are available.
2. Initialize  $j = 1$ .
3. Draw the  $1 + 2n$  sigma-points  $\mathbf{x}(t_{k+N-j})^{(i)}$  using  $\mathbf{x}(t_{k+N-j})$  as mean value, and  $P_{k+N-j}$  as covariance matrix.
4. Propagate the sigma points through the dynamic model (prediction step) and compute the mean, covariance and cross covariance

$$\begin{aligned} \mathbf{x}(t_{k+N-j+1})^{(i)} &= f(\mathbf{x}(t_{k+N-j})^{(i)}, \mathbf{u}(t_{k+N-j}), \Theta(t_{k+N-j}), t_{k+N-j}), i = 0 \dots 2n + 1, \\ \mu_{k+N-j+1}^- &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{x}(t_{k+N-j+1})^{(i)}, \\ P_{k+N-j+1}^- &= P_k + \sum_{i=0}^{2n+1} w_c^{(i)} \left( \mathbf{x}(t_{k+N-j+1})^{(i)} - \mu_{k+N-j+1}^- \right) \left( \mathbf{x}(t_{k+N-j+1})^{(i)} - \mu_{k+N-j+1}^- \right)^T, \\ C_{k+N-j+1} &= \sum_{i=0}^{2n+1} w_c^{(i)} \left( \mathbf{x}(t_{k+N-j+1})^{(i)} - \mu_{k+N-j+1}^- \right) \left( \mathbf{x}(t_{k+N-j})^{(i)} - \mathbf{x}(t_{k+N-j}) \right)^T. \end{aligned}$$

5. Compute the smoother gain, and post-correct the previous estimation providing the smoothed mean and covariance

$$\begin{aligned} K &= C_{k+N-j+1} \left[ P_{k+N-j+1}^- \right]^{-1}, \\ \mu_{k+N-j}^s &= \mu_{k+N-j+1}^- + K \left[ \mathbf{x}(t_{k+N-j+1}) - \mu_{k+N-j+1}^- \right], \\ P_{k+N-j}^s &= P_{k+N-j} + K \left[ P_{k+N-j+1}^s - P_{k+N-j+1}^- \right] K^T. \end{aligned}$$

6. If  $j < N$  then set  $j := j + 1$  and go to step (2), otherwise exit.

N.B. At the end of each time step the state is estimated as  $\hat{\mathbf{x}}^s(t_{k+N-j}) \sim N(\mu_{k+N-j}^s, P_{k+N-j}^s)$ .

The first barrier to overcome was that the state estimation procedure refers to the full vector of state variables  $\mathbf{x}(\cdot)$ . However in many applications, the number of state variables is higher than the ones to be estimated. An example are sensors that sometimes contain a unitary gain first order filter. Including all the state variables in the estimation is not desirable because

- the number of sigma points required by the UKF, and thus the number simulations to be run, grows with the number of state variables.

- for every state variable or parameter estimated the user must provide an initial guess for mean value and the covariance.

Therefore, the user is allowed to select a subset of the state variables and parameters to be estimated by the UKF algorithm.

Similarly, an FMU may have several outputs, but only few of them may be measured and used by the UKF. The outputs for which a measurement is available and used by the UKF are denominated measured

outputs. The measured outputs requires additional information such the covariance (i.e., the uncertainty associated to the measure) and a time serie containing the data.

All the inputs and the measured outputs have to be associated with a time serie. This can be done by associating them with a specific column of a csv file.

Another issue has been encountered regarding the range of validity of states and parameters. For example, the level of water contained in a tank has to be positive but not greater than the height of the tank. We addressed this issue by constraining the sigma points within user specified upper and lower limits. As default, the min and max values specified in the FMU model description file are used.

We also implemented the ability to run the simulations in parallel. The computationally demanding part of the UKF and Smoother is the time integration done using PyFMI. However this section of the algorithm is entirely parallelizable. By default, our implementation uses one less thread as there are processors to run the simulation, while a thread manages the simulation and collects the results. The simulation results generated by PyFMI can optionally be written to files or not.

All these functionalities have been embedded in few classes that use PyFMI. In particular we have defined a new class representing the FMU model in a more general term since the state estimation is a task that involves more than a simple simulation, which is the aim of PyFMI.

## 2.6 Code snippet

This subsection contains a code snippet that illustrates how the FMU-based state and parameter estimation framework works. In this example the FMU model represents a mass-spring-damper system, where the input is the force  $F$  applied to the mass, and the measured output is the mass acceleration  $a$ . The two corresponding data series are stored in a csv file named data.csv. The states to be estimated are the position  $x$  and its velocity  $v$ .

```
# Path of the FMU model
filePath = "./model.fmu"

# Instantiate the model
m = Model(filePath, atol=1e-5, rtol=1e-4)

# Path of the CSV file that contains the data series
csvPath = "./data.csv"

# Associate the columns of the csv file to the input
input = m.GetInputByName("F")
input.GetCsvReader().OpenCsv(csvPath)
input.GetCsvReader().SetSelectedColumn("Force")

# Associate the columns of the csv file to the output
output = m.GetOutputByName("a")
output.GetCsvReader().OpenCsv(csvPath)
output.GetCsvReader().SetSelectedColumn("acceleration")
```

```
# Specify that this output has to be compared against
# measured data contained in the CSV file
output.SetMeasuredOutput()

# Specify output measurement covariance
output.SetCovariance(1.0)

# Specify the subset of the states to be estimated
m.AddVariable(m.GetVariableObject("x"))
m.AddVariable(m.GetVariableObject("v"))

# Get a reference to the states to be estimated
var_x = m.GetVariables()[0]
var_v = m.GetVariables()[1]

# Specify initial value for the position
# and the boundary limits (e.g., position x must be positive)
var_x.SetInitialValue(2.5)
var_x.SetCovariance(0.5)
var_x.SetMinValue(0.0)

# Specify initial value for the velocity
var_y.SetInitialValue(0.0)
var_y.SetCovariance(0.2)

# Initialize simulator
m.InitializeSimulator()

# Instantiate UKF and pass to it the model
UKF = ukfFMU(m)

# Run the filter from 0.0 to 10.0 seconds
time, X, Sx, y, Sy = UKF.filter(start = 0.0, stop = 10.0)

# plotting ...
```

This framework reduces the effort to set up a state or parameter estimation. The model can be created in Modelica and directly imported as an FMU, avoiding the user to rewrite the model in the right format required by the UKF. Other functionalities provide an easy way to specify the state variables or parameter to be estimated, together with the data series to be used as inputs and outputs.

## 3 Application: FDD

This section contains an example that shows how the FMU-based state and parameter estimation algorithm can be used for fault detection and diagnosis.

FDD algorithms based on state estimation techniques are known to be more suitable than other approaches based on neural networks or principal component analysis for detecting multiple faults [9, 10, 11], they can compute the fault probabilities and they provide an indication of what is not working as expected in the system. All these features are possible thank to the probabilistic description of the state variables and parameters the estimation techniques provide.

A drawback of state estimation based strategies is that they require a slightly higher modeling effort, e.g., to include explicit fault descriptions in the model itself. However, various open-source modeling tools (e.g. OpenModelica and JModelica), and modeling libraries for buildings (Modelica Buildings library [12]) are available, and they provide two main advantages: reducing the effort required to set up the model, and

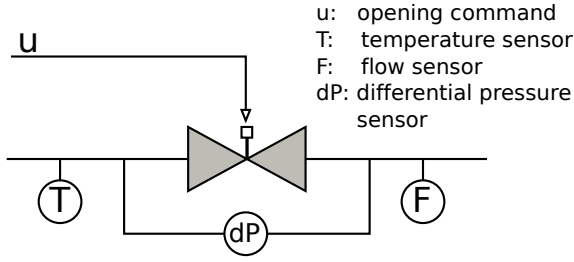


Figure 3: Schematic of the system.

reducing the risk of modeling errors since they are typically validated and tested.

The availability of these tools and libraries together with the FMU-based state and parameter estimation algorithm put in place a framework for creating with a low level of effort a model-based FDD algorithm.

The considered system is a valve that regulates the water flow rate in a water distribution system (see Figure 3). The system is described by the following equations

$$\dot{m}(t) = \phi(x(t))A_v\sqrt{\rho(t)}\sqrt{\Delta p(t)}, \quad (8a)$$

$$x(t) + \tau\dot{x}(t) = u(t), \quad (8b)$$

where  $\dot{m}(\cdot)$  is the mass flow rate passing through the valve,  $\Delta p(\cdot)$  is the pressure difference across it,  $u(\cdot)$  is the valve opening command signal,  $x(\cdot)$  is the valve opening position,  $\tau$  is the actuator time constant,  $\phi(\cdot)$  is the power-law opening characteristic,  $A_v$  is the flow coefficient and  $\rho(\cdot)$  is the fluid density (please note that the square root of the pressure difference is regularized around zero flow in order to prevent singularities in the solution). The system has three sensors (see Figure 3) that respectively measure the pressure difference across the valve, the water temperature  $T(\cdot)$  and the mass flow rate passing through it. All the sensors are affected by measurement noise. In addition, the mass flow rate sensor is also affected by a thermal drift.

$$T^N(t) = T(t) + \eta_T(t) \quad (9a)$$

$$\Delta p^N(t) = \Delta p(t) + \eta_P(t) \quad (9b)$$

$$\dot{m}^{N+D}(t) = (1 + \lambda(T(t) - T_{ref}))\dot{m}(t) + \eta_m(t) \quad (9c)$$

The measurement equations are described in (9), where the superscript  $N$  indicates a measurement affected by noise, the superscript  $N+D$  indicates the presence of both noise and thermal drift,  $T_{ref}$  is the reference temperature at which the sensor has no drift,  $\lambda$  is the thermal drift coefficient and  $\eta_T(\cdot)$ ,  $\eta_P(\cdot)$ , and  $\eta_m(\cdot)$  are three uniform white noises affecting respectively the temperature, pressure and mass flow rate

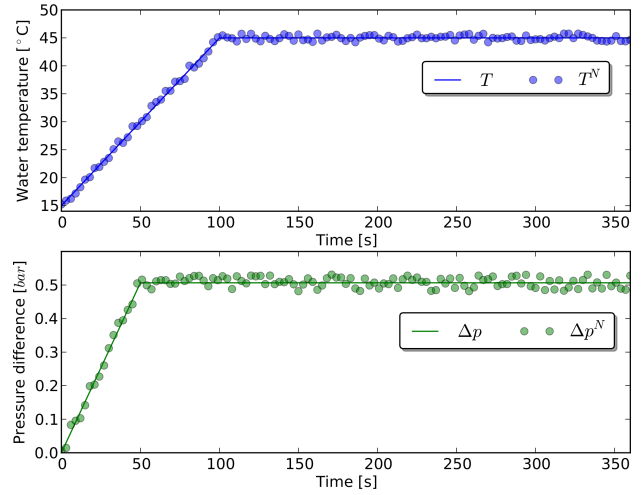


Figure 4: Input signals of the faulty valve model: water temperature (blue) and pressure difference (green). The lines represent the data generated by simulation, while the dots represent the sampled and noisy version provided to the UKF.

measurements. These signals are sampled every two seconds.

Suppose during the operation, at  $t = 80$  s, the valve becomes faulty. The fault affects the ability of the valve to control its opening position. The valve opening cannot go below 20% (causing a leakage) and over 60% (it gets stuck). At  $t = 250$  s, the valve stuck position moves from 60% to 90%.

The fault identification procedure is asked to identify whether the valve not works as expected, that is its opening position follows the command signal. The fault identification is performed using the UKF that uses as input signals for its model the noisy pressure difference (see Figure 4), the noisy water temperature (see Figure 4) and the command signal (see Figure 6). The command signal is noise free because it is computed by some external controller and not measured. The UKF compares the output of its simulations with the measured mass flow rate (see Figure 5) that is affected by both noise and thermal drift. The effect of the thermal drift is visible in Figure 5 where the green dots represent the measured mass flow rate while the green line is the actual mass flow rate passing through the valve.

The UKF and the smoother estimate the value of the state variable  $x(t)$  representing the valve opening position and the parameter  $\lambda$ , the thermal drift coefficient. The length of the augmented state is  $n = 2$ . Hence for every estimation step, the UKF performs  $1 + 2 \times 2 = 5$  simulations. The UKF has the initial conditions  $x(0) \sim N(0.8, 0.05)$  and  $\lambda \sim N(0, 0.7 \cdot 10^{-3})$ , the output noise

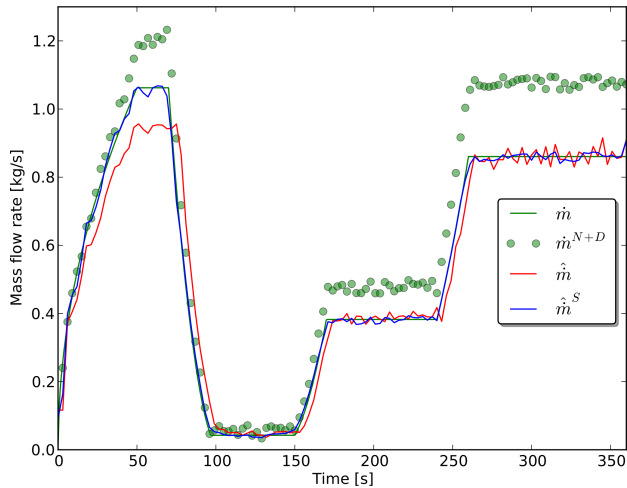


Figure 5: The green line represent the mass flow rate that is passing through the valve. The green dots are the measurements of the mass flow rate (affected by noise and sensor thermal drift) used by the UKF. The red line is the UKF estimation, while the blue line is the smoother estimation.

covariance matrix is  $R = [0.05]$ , and the filter coefficients are  $\alpha = \frac{1}{\sqrt{3}}$ ,  $\beta = 2$  and  $k = 3 - n = 1$ .

As shown in Figure 5, the measured mass flow rate (green dots) are far from the real mass flow rate (green line). Despite the measurement error, the UKF and the smoother provide an estimation with a good accuracy (red and blue lines in Figure 5). The mass flow rates computed by the UKF,  $\hat{m}$ , and the smoother,  $\hat{m}^S$ , are close to the real one,  $\dot{m}$ , because they are able to estimate both the valve opening position and the sensor drift coefficient, as shown in Figures 6 and 7. As expected the smoother is able to provide a better estimation since it uses more data. The time spent by the UKF to perform the simulations and computing the estimations was about 0.05 s, that is lower than the sampling time of 2 s. The speed of this UKF based FDD algorithm allows a real-time implementation for this particular application.

## 4 General Applicability

The UKF and in general state/parameter estimation techniques are well known solutions used in many fields. We presented an approach that reduces the effort needed to set up a state and parameter estimation for models created with simulation programs that are FMI compliant. The example shows how this tool can be used for FDD purposes in the context of HVAC systems. However, this tool can be used in other con-

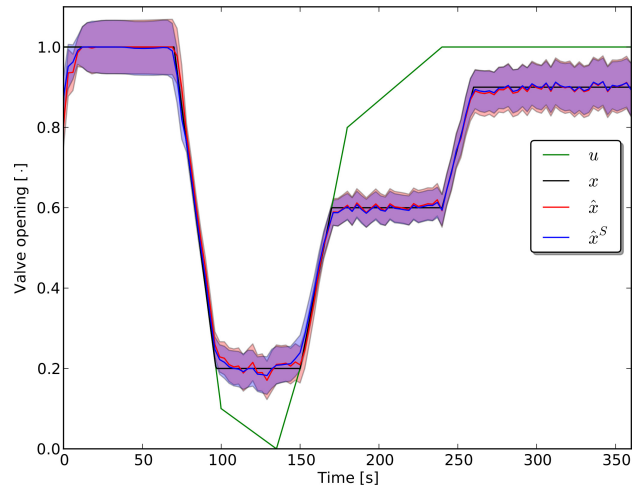


Figure 6: The green line is the opening valve signal. The blue line is the actual opening value affected by faults. The red and blue lines are the UKF and smoother estimations of the valve opening position (the area surrounding the estimation is the  $\sigma$ -confidence interval).

texts. For example, state and parameter estimation techniques are used to support guidance and control systems in the automotive, robotic and aerospace industries.

## 5 Conclusion

We proposed a model-based state and parameter estimator for dynamic systems described using the FMI standard. The paper explained the nonlinear state estimation and smoothing techniques employed by the algorithm, together with the details necessary to implement it. The last section shows how the FMU-based state and parameter estimator can be used to set up a fault detection algorithm capable of identifying faults in a valve, even in presence of wrong and noisy measurements.

This algorithm extends the functionalities of any simulation program that implements the FMI standard version 1.0. As shown in the example, this algorithm has a direct application in FDD, but it can also be used for model calibration and process control together with adaptive or model predictive control schemes. The main advantage of this algorithm is that it allows to reuse models from the design phase, it extends the capabilities of FMI compliant modeling frameworks and it reduces the time and investments necessary to develop advanced control strategies, calibration and FDD.

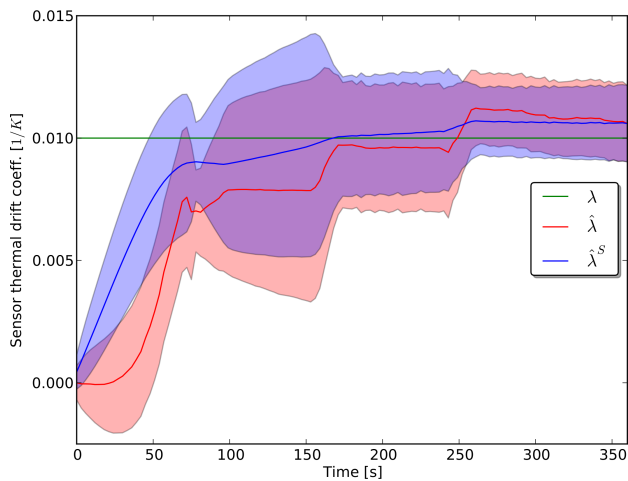


Figure 7: The green line is the sensor thermal drift coefficient, while the red and blue lines are the estimations of the thermal drift coefficient computed by the UKF and smoother (the area surrounding the estimation is the  $\sigma$ -confidence interval).

## 6 Acknowledgements

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231. The research was also supported by the U.S. Department of Defense under the ESTCP program.

The authors thank Mary Ann Piette, Jessica Granderson, Oren Shetrit, Wangda Zuo, and Rong Lily Hu for the support provided through the project.

## References

- [1] S. J. Julier and J. K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. *Robotics Research Group Technical Report, Department of Engineering Science, University of Oxford*, pages 1–27, November 1996.
- [2] S.J. Julier. The scaled unscented transformation. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 4555–4559 vol.6, 2002.
- [3] Simon S Haykin et al. *Kalman filtering and neural networks*. Wiley Online Library, 2001.
- [4] D. Crisan and Arnaud Doucet. A survey of convergence results on particle filtering methods for practitioners. *Signal Processing, IEEE Transactions on*, 50(3):736–746, 2002.
- [5] Modelon AB. PyFMI a package for working with dynamic models compliant with the functional mock-up interface standard, September 2013.
- [6] E.A. Wan and R. Van der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158, 2000.
- [7] S. Sarkka. On unscented kalman filtering for state estimation of continuous-time nonlinear systems. *Automatic Control, IEEE Transactions on*, 52(9):1631–1641, 2007.
- [8] S. Sarkka. Unscented rauch–tung–striebel smoother. *Automatic Control, IEEE Transactions on*, 53(3):845–849, 2008.
- [9] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293 – 311, 2003.
- [10] Venkat Venkatasubramanian, Raghunathan Rengaswamy, and Surya N Kavuri. A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies. *Computers & Chemical Engineering*, 27(3):313 – 326, 2003.
- [11] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Surya N. Kavuri, and Kewen Yin. A review of process fault detection and diagnosis: Part III: Process history based methods. *Computers & Chemical Engineering*, 27(3):327 – 346, 2003.
- [12] Michael Wetter, Wangda Zuo, Thierry S Noidui, and Xiufeng Pang. Modelica buildings library. *Journal of Building Performance Simulation*, (In press):1–18, 2013.



# Grey-Box Building Models for Model Order Reduction and Control

Roel De Coninck<sup>a,b</sup>, Fredrik Magnusson<sup>c</sup>, Johan Åkesson<sup>c,d</sup>, Lieve Helsen<sup>b</sup>

<sup>a</sup>3E nv., 1000 Brussels, Belgium,

<sup>b</sup>KU Leuven, Department of Mechanical Engineering, 3001 Heverlee, Belgium,

<sup>c</sup>Department of Automatic Control, Lund University, SE-221 00 Lund, Sweden,

<sup>d</sup>Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

## Abstract

As automatic sensing and Information and Communication Technology (ICT) get cheaper, building monitoring data is easier to obtain. The abundance of data leads to new opportunities in the context of energy efficiency in buildings. This paper describes ongoing developments and first results of data-driven grey-box modelling for buildings. A Python toolbox is developed based on a Modelica library with thermal building and Heating, Ventilation and Air-Conditioning (HVAC) models and the optimisation framework in JModelica.org. The tool chain facilitates and automates the different steps in the system identification procedure, like data handling, model selection, parameter estimation and validation. The results of a system identification and parameter estimation for a single-family dwelling are presented.

*Keywords: buildings, grey-box models, parameter estimation, collocation method*

## 1 Introduction

The continuous progress in ICT has led to the availability of small and low-cost sensors, low power wireless data transfer protocols, cheap and accessible data storage and powerful servers. Applied to the building sector, these technologies can be used to collect massive amounts of building monitoring data at relatively low costs. The abundance of data gives rise to new opportunities and applications in existing buildings like fault detection, energy efficiency analysis and model-based building operation. A first step in many of these applications is the creation of a building energy model.

Building models can be classified in three categories: white-box, grey-box and black-box models [1–3]. White-box modelling bases the model solely on

prior physical knowledge of the building. Most building simulation software falls under this category, like TRNSYS, EnergyPlus and many others [4]. Black-box modelling bases the model solely on response data (monitoring of the building) and a universal model set, including e.g. AR and ARMAX. No physical insight is required for making a black-box model. Grey-box identification methods and tools cater for the situation where prior knowledge of the object is not comprehensive enough for satisfactory white-box modelling and, in addition, purely empirical black-box methods do not suffice because the involved physical processes are too complex.

The difference between white- and grey-box modelling is not in the complexity of the model. A single-state model can be a white-box model if all parameters can be fixed based on physical knowledge only. However, when one or more parameters in a white-box model are estimated based on a fitting of the model to measurement data, the model becomes grey, no matter its complexity. Therefore, the distinction between white and grey cannot be made by only looking at the model structure; one has to know how the model parameters have been identified.

All three model types can be either deterministic or stochastic. A deterministic model cannot explain the differences between the model output and the true variations of the states (observations). Madsen and Holst [2] therefore introduced a Wiener process in the system equations to cope with the simplifications of the model and uncertainties in inputs and monitoring. The obtained model is a stochastic state-space model.

For existing buildings with available monitoring data, the grey-box approach is considered to combine the best of two worlds: physical insight and model structure from the white-box paradigm and parameter estimation and statistical framework from the black-box paradigm. This paper describes an approach to

grey-box modelling for buildings and the development of a toolbox combining Modelica and Python. The resulting framework will be referred to as *the toolbox* in the remainder of this paper.

The aim of the toolbox is to identify low-order models from (limited) building monitoring datasets. When the dataset is generated by a detailed building simulation model instead of an existing building, we speak of model order reduction. The obtained models can be used in order to set up Model Predictive Control (MPC) or to scale up simulations from single buildings to neighbourhoods and districts.

This paper describes the methodology of the toolbox and presents some results of the application to a model order reduction of a single-family dwelling.

## 2 Methodology

### 2.1 Overview

A high-level overview of the toolbox is shown in Figure 1. The toolbox is composed of four major components:

1. Modelica library *FastBuildings* with thermal zone models, HVAC components and building models;
2. different *.mop files* specifying the model components and which parameters to estimate;
3. *JModelica.org* as a middle layer for compilation of the *.mop files* as well as formulation and solution of the optimisation problem;
4. Python module *greybox.py* delivering the user interface and top-level functionality.

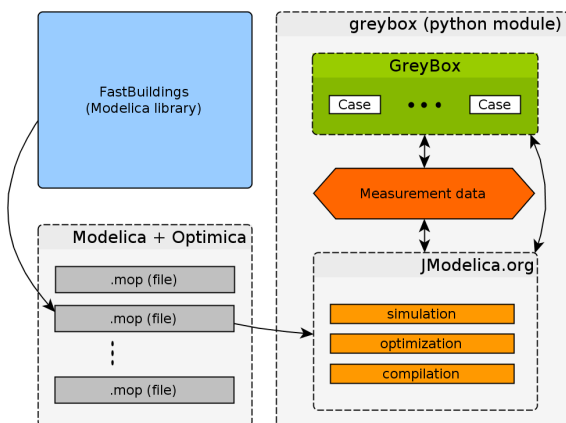


Figure 1: Overview of the grey-box buildings toolbox

### 2.2 Modelica

Modelica is gaining importance in the building simulation community [5, 6]. The choice of Modelica for the construction of the models is based on two major arguments. First, Modelica allows for linear, non-linear and hybrid model formulations and therefore it does not limit the model structure as such. Second, Modelica is equation-based, thus allowing efficient Newton-type solvers to be used as an alternative to for example genetic algorithms. Moreover, as shown in Section 3, the interfaces of the low-order models are identical to the detailed building model used in the *IDEAS* library [7], enabling easy model exchange.

### 2.3 Models

Every model structure for which the parameters have to be estimated is characterized by a different *.mop* file, of which the format is very similar to an ordinary Modelica (*.mo*) file. The *.mop* file extension is specified by JModelica.org, which is described in Section 2.4. Each *.mop* file has the same structure and has to define two models: one model for simulation, called *Sim*, and one for parameter estimation, called *ParEst*. By default, the models are based on the *FastBuildings* Modelica library, which has been developed in conjunction with this toolbox. However, this is not required for the toolbox to work, as long as some naming conventions are followed. The *FastBuildings* library is introduced in Section 3. Any parameter present in the model can be estimated, including initial values of the states.

### 2.4 The JModelica.org platform

The toolbox relies heavily on the JModelica.org [8] platform, which is an open-source tool for modelling, simulation and optimisation of dynamic systems described by Modelica code. For simulation purposes, JModelica.org relies on the Functional Mockup Interface [9]. For optimisation purposes, JModelica.org offers various algorithms and also supports the Modelica language extension *Optimica*. *Optimica* allows for high-level formulation of dynamic optimisation problems of the type presented in Section 2.5. The file format *.mop* is used for *Optimica* code.

The optimization algorithm used by the toolbox to estimate the parameters is collocation-based and is presented in Section 2.6 and described in more detail in [10], where in particular optimal control is also treated. IPOPT [11], built with the MA27 solver of HSL [12], is used to solve the non-linear program (NLP) that arises from the collocation method.

Since IPOPT uses a gradient-based method, first- and second-order derivatives of all the expressions in the NLP with respect to all of the decision variables are needed. To this end, CasADi [13] is used to construct the NLP and then to compute the needed derivatives by algorithmic differentiation.

## 2.5 Problem formulation

Identification of the unknown model parameters is formulated as a dynamic optimization problem of the general form

$$\text{minimize} \quad \int_{t_0}^{t_f} e(t)^T \cdot Q \cdot e(t) dt, \quad (1a)$$

$$\text{with respect to} \quad \dot{x}, x, w, u, p,$$

$$\text{subject to} \quad F(t, \dot{x}(t), x(t), w(t), u(t), p) = 0, \quad (1b)$$

$$x(t_0) = x_0, \quad (1c)$$

$$\forall t \in [t_0, t_f].$$

The system dynamics are modelled by a single, possibly implicit, non-linear and time-variant, differential-algebraic equation (DAE) system of at most index one. That is, an equation system of the form (1b), where  $t$  is the time,  $x$  is the state,  $w$  is the vector-valued algebraic variable,  $u$  is the vector-valued system input, which includes both control variables and known disturbances, and  $p$  is the vector of parameters to be estimated. Since a gradient-based method will be applied to solve the dynamic optimization problem,  $F$  needs to be twice continuously differentiable with respect to all of its arguments except the first one (time). This disables the use of hybrid models. Initial conditions are given by specifying the initial state, that is, on the form of (1c), where  $t_0$  is the start time. The initial state is usually unknown, in which case some, or all, elements of  $x_0$  can also be introduced as elements of the vector  $p$ .

The objective (1a) of the optimisation is to minimise the integrated quadratic deviation  $e$  of the model output from the corresponding measurement data. The model output  $y$  is typically some of the states, but could also be some of the algebraic variables (and also inputs, as discussed below). The matrix  $Q$ , which typically is diagonal, is used to weight the different outputs. The measurement data is assumed to be a function of time, denoted by  $y_m$ . Since measurements are typically discrete in time, they are simply interpolated linearly to form  $y_m$ . The output deviation  $e$  is then given by

$$e(t) := y(t) - y_m(t). \quad (2)$$

The inputs can be treated in two different ways. The first is to assume that the inputs are known exactly by their measurement data and eliminate them from the problem. The second way is to have an error-in-variables approach where the inputs are kept as decision variables and treat them as model output, that is, include them in the vector  $y$  and penalize their deviation from the corresponding measurement data. The second way is useful for coping with uncertainties in the measurement data.

## 2.6 Solution algorithm

The approach taken to solve (1) is based on low-order direct collocation, see [14]. The idea is to divide the time horizon into a number of elements,  $n_e$ , and approximate the time-variant system variables  $\dot{x}, x, w$  and  $u$  by a polynomial of time within each element, called a collocation polynomial. These polynomials are determined by enforcing the dynamic constraints at a certain number of points,  $n_c$ , within each element. These points are called collocation points and  $t_{i,k}$  is used to denote collocation point number  $k \in [1..n_c]$ , where  $[1..n_c]$  denotes the integer interval between 1 and  $n_c$ , in element number  $i \in [1..n_e]$ . The system variables' values at these points, denoted by

$$(\dot{x}_{i,k}, x_{i,k}, w_{i,k}, u_{i,k}, e_{i,k}) :=$$

$$(\dot{x}(t_{i,k}), x(t_{i,k}), w(t_{i,k}), u(t_{i,k}), e(t_{i,k})),$$

are then interpolated based on Lagrange interpolation polynomials to form the collocation polynomials. There are different schemes for choosing the placement of collocation points with different numerical properties. In this paper we only consider Radau collocation. All collocation methods correspond to special cases of implicit Runge-Kutta methods and thus inherit desirable stability properties making them suitable for stiff systems.

This approximation reduces the Problem (1), which is of infinite dimension, into a finite-dimensional non-

linear program of the form

$$\min. \sum_{i=1}^{n_e} \left( h_i \cdot \sum_{k=1}^{n_c} \omega_k \cdot e_{i,k}^T \cdot Q \cdot e_{i,k} \right), \quad (3a)$$

$$\text{w.r.t. } \dot{x}_{i,k}, x_{i,l}, w_{i,k}, u_{i,k}, p,$$

$$\text{s.t. } F(t_{i,k}, \dot{x}_{i,k}, x_{i,k}, w_{i,k}, u_{i,k}, p) = 0, \quad (3b)$$

$$x_{1,0} = x_0, \quad (3c)$$

$$x_{n,n_c} = x_{n+1,0}, \quad \forall n \in [1..n_e - 1], \quad (3d)$$

$$\dot{x}_{i,k} = \frac{1}{h_i} \sum_{l=0}^{n_c} \alpha_{l,k} \cdot x_{i,l}, \quad (3e)$$

$$\forall (i, k, l) \in ([1..n_e] \times [1..n_c] \times [0..n_c]).$$

The NLP objective (3a) is an approximation of the original objective (1a) based on a Gaussian-like quadrature formula, where the measurement error  $e_{i,k}$  in each collocation point is summed and weighted by the corresponding element length  $h_i$ , which is fixed a priori, and the quadrature weight  $\omega_k$ , which depends on the choice of collocation points. Notice that the decision variables are not only the unknown parameters  $p$ , but also the discretized system variables  $\dot{x}_{i,k}, x_{i,l}, w_{i,k}$  and  $u_{i,k}$ .

Since the states need to be continuous (but not differentiable) with respect to time, the new continuity constraint (3d) needs to be introduced. Because we use Radau collocation, where no collocation point exists at the start of each element, this also requires the introduction of the new variables  $x_{i,0}$ , which represent the value of the state at the start of element  $i$ . With the introduction of  $x_{1,0}$ , the initial condition (1c) is straightforward to transcribe into (3c). The dynamic constraint (1b) is also straightforward to transcribe into (3b), by only enforcing it at the collocation points instead of during the entire time horizon.

Finally, we introduce the constraints (3e) to capture the dependency between  $x$  and  $\dot{x}$ . The state derivative  $\dot{x}_{i,k}$  in a collocation point is approximated by a finite difference of the collocation point values of the state in that element. The finite difference weights  $\alpha_{l,k}$  are related to the butcher tableau of the Runge-Kutta method that corresponds to the collocation method.

All that remains is to solve the NLP (3) in order to obtain an approximate solution to the original Problem (1). This can be done using dedicated NLP software; in our case IPOPT.

## 2.7 Toolbox functionality and workflow

The user interacts with the toolbox through the *greybox.py* Python module. This module defines two classes *GreyBox* and *Case*, as shown in Figure 1.

The idea is to instantiate the *GreyBox* class once for the system identification of a given building. The *GreyBox* object will contain many different instances of the *Case* class. Every *Case* is an attempt (successful or not) to obtain a model for the given building. The *Case* therefore keeps track of the model structure, identification data, initial guess, solver settings and results of a single attempt. The functionality of the toolbox is packed in methods of the *GreyBox* class and can be grouped into different domains, according to the foreseen workflow. This is shown in Figure 2. This workflow is discussed in the following paragraphs.

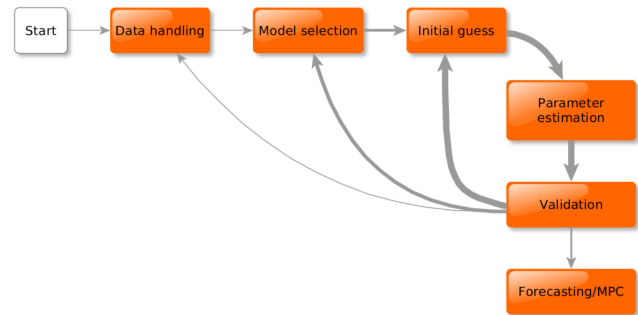


Figure 2: Workflow and high-level functionality in *greybox.py*

The methods under **data handling** are used to load the data files, resample the data if desired, create data slices of given lengths (for example one week, but can be any period) and show a plot of any data slice. Typically, one data slice is the training set, and the other slices can be used for cross-validation.

When the data has been pre-processed, a model structure has to be specified in the **model selection** step. This is accomplished by specifying the path to a *.mop* file. There are two models in the *.mop* file: a Modelica model for simulation, called *Sim*, and a Modelica + Optimica model for parameter estimation, called *ParEst*. The main difference is that in the model *ParEst*, the value of the Optimica attribute `free` is set to `true` for each parameter to be estimated. The compilation of both models happens automatically by invoking the corresponding *JModelica.org* functionality. This includes extracting information from the model (state vector, parameter vector and required inputs) and getting solver options.

Before the parameter estimation can be started, an **initial guess** has to be specified for each element in the parameter vector. These can be obtained by default, by inheritance, by Latin hypercube sampling or manually.

When the default initial guesses are used, an appropriate value is chosen for each parameter, based on its

name. For example, the naming convention in *FastBuildings* forces all parameter names for thermal resistances to start with 'r' (like `rWa1`), for thermal capacities with 'c' (like `cZon`), for fractions with 'fra' (like `fraRad`), etc. Based on the first letter(s) of a parameter to be estimated, a default initial value can be set.

An alternative for obtaining the initial guess is to start from the optimized parameter vector from a previous case. This is especially useful when a new *.mop* file is selected that has similarities with a previously processed *.mop* file. Due to the naming conventions in *FastBuildings*, the corresponding parameters will have the same name. Therefore, the best initial guess for the parameters in the new model will be the optimal value from the previous estimation. For new parameters, the default initial guess method described above is used.

The last automated option to obtain initial guesses is based on Latin hypercube sampling. Due to the non-convexity of the problem, there can potentially exist many local minima. To investigate the parameter search space more systematically and increase the chances of finding a global minimum, a Latin hypercube sampling method has been implemented. This method will take a single initial guess as well as lower and upper bounds for each parameter and derive a univariate beta distribution from these three values. The distribution can be symmetric or asymmetric, as shown in Figure 3.

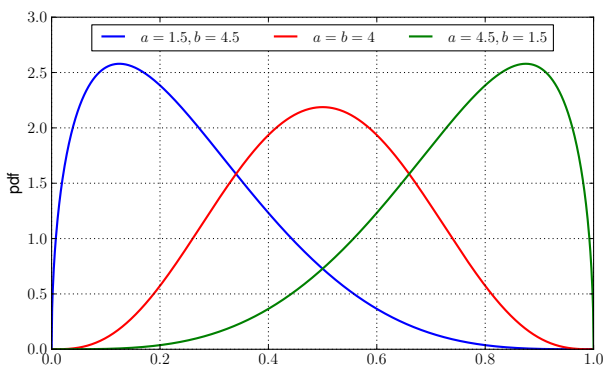


Figure 3: Symmetric and asymmetric beta distributions

The Latin hypercube sampling will then derive  $n$  stratified samples from each distribution and combine them randomly to obtain  $n$  different initial guesses. Each of these guesses will be copied to a new case to keep track of the results.

When a case has an initial guess for the parame-

ter vector, the **parameter estimation** can be started. However, the NLP described in Equation (3) requires good initial guesses for each of the decision variables in the collocation problem (including all collocated states and algebraic variables). This is handled by doing a simulation first with the `Sim` model and the initial guess of the parameter vector. The resulting simulation trajectories are used as initial guesses for the decision variables in Problem (3). Numerical scaling factors for each system variable are also computed as the infinity norm of the corresponding trajectory.

The solution time and the number of iterations can vary a lot depending on the initial guess and the ability of the model to represent the measurement data. IPOPT allows the specification of a maximum solution time and/or a maximum number of iterations after which it will interrupt the optimisation. Both can be set in the toolbox, but in practice, specifying a maximum solution time is more intuitive. It is also more effective when the iterations become very slow. Moreover, experience shows that long execution times often lead to solutions far away from the global optimum or even divergence.

The estimation will add the optimized parameter vector to the case, as well as the IPOPT solver statistics.

The **validation** of the results is always based on a post-simulation with the `Sim` model and the optimized values of the parameter vector. There are both visual and quantitative validation methods. The visual methods contain for example time series plots of the resulting trajectories and corresponding residuals, scatter plots of the residuals with monitoring data and a plot of the autocorrelation function of the residuals. This also implies a check on the weights of the matrix  $Q$  from (1a) in case the error-in-variables method is used. When a full Latin hypercube sample has been estimated, a visual check of the different local optima is implemented. This can whether the sample was large enough to suppose the global optimum to be found. The quantitative methods are based on a computation of the root-mean-square error (RMSE) for each trajectory in the vector  $e$  from Equation (2). This can be done on the training data (auto-validation) or on any other dataset (cross-validation). As the RMSE is computed based on the post-simulation, possible discretisation errors in the collocation method are disregarded in the model validation process.

A computation of the confidence interval for each of the estimated parameters is implemented. This will give an indication of the accuracy of the estimation and

the parameter's influence on the model's input-output behaviour. The standard deviation of the estimated parameters  $\hat{p}$  is computed according to Englezos and Kalogerakis [15]. The standard deviation for parameter  $i$  is the square root of the diagonal element on  $(i, i)$  in the covariance matrix  $\text{cov}(\hat{p})$  of the estimated parameters, which is given by

$$\text{cov}(p^*) = \hat{\sigma}^2 (J^T J)^{-1},$$

where  $J$  is the Jacobian of the trajectories with respect to the estimated parameters and  $\hat{\sigma}^2$  is the estimated variance of the output deviation  $e$ .

The final step in the system identification is **model selection**. Model selection should be carried out on two levels: for a single model, and between different models. For a single model, generally a Latin hypercube sampling is executed and the resulting global optimum is taken if it is a valid solution. Valid means that:

- the parameters do not lie on the specified minimum or maximum bounds,
- the parameter values are physically reasonable,
- the confidence intervals are within reasonable bounds.

The normal procedure for an inter-model selection procedure starts by a parameter estimation on a very simple (first-order) model. The obtained parameter values are often a good indication of the order of magnitude of the parameters for the more detailed models later on. Then, different models of higher order and complexity are estimated, and only those for which the single model validation is satisfactory are retained. Among all the retained models, the best model is the one that leads to the lowest RMSE value for cross-validation. This approach avoids overfitting of the model, as will be demonstrated in Section 4. A more focused forward selection procedure as described by Bacher and Madsen [16] can also be applied.

### 3 FastBuildings library

The *FastBuildings* library targets low-order building modelling. The library has sub-packages for thermal zone models (including windows), HVAC, user behaviour, inputs, buildings and examples. Single and multi-zone building models can be created easily by instantiating one of the predefined templates in the *Building* sub-package and redeclaring the desired

submodels, like the thermal zone, HVAC or window model. The following design principles are applied throughout the library.

- The thermal connectors are `HeatPorts` from the `Modelica.Thermal` package.
- Thermal resistors and capacitances are not used from the *Modelica Standard Library* (MSL). Simplified versions with less auxiliary variables are implemented. They have exactly the same interface and connectors for compatibility with the MSL.
- A strict naming convention is used for consistency and to enable the *greybox.py* toolbox to automate certain tasks.
- The library heavily relies on the `extends` construct in order to avoid code duplication. This is specifically useful for the thermal zone models that have increasing complexity as a function of their order.
- An inner/outer component `simFasBui` passes all inputs like weather data, occupancy etc. from the top most level to all sublevels.
- The models for thermal zones, HVAC and user behaviour have exactly the same interface as their equivalents in the *IDEAS* library [7]. Therefore, it is very easy to replace one or more detailed models from an *IDEAS*-based model by a low-order equivalent from the *FastBuildings* library.

Currently, the thermal zone models available in the library are based on a resistor-capacitance (RC) network analogy which is often used for the modelling of thermal processes. This is however not required; any model that specifies a relationship between the heat flows and temperatures at the interface of a thermal zone can be implemented. An example of one of the third-order models in the library is given in Figure 4.

The library is distributed with the *Modelica license 2* and can be found in the *open-IDEAS* source code repository on Github [17].

### 4 Results

The toolbox is applied on a case study for model order reduction on a single family dwelling. A low-order model is derived from the simulated trajectories that are obtained with a detailed model. Prior knowledge about the dwelling is not taken into account in the

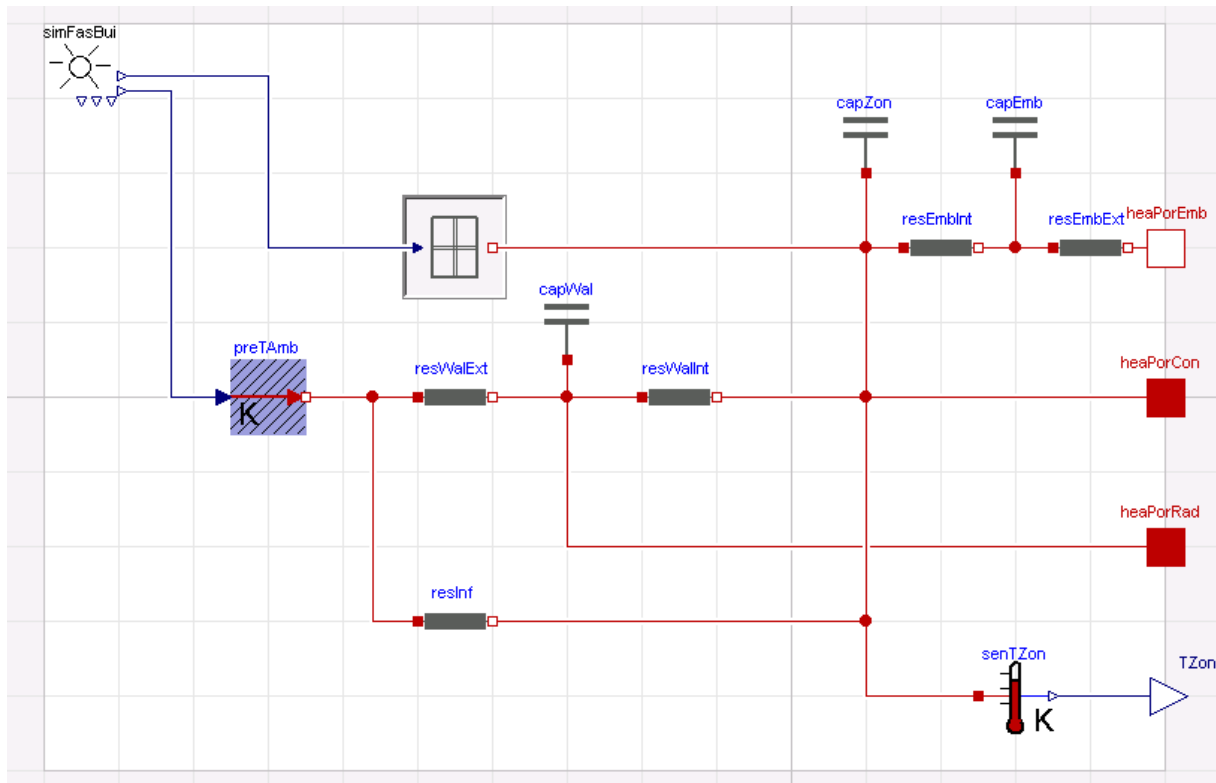


Figure 4: Example of a third-order thermal zone model in the FastBuildings library

model selection or parameter estimation, but is given here. The dwelling has two construction layers with a total heated floor surface of  $196 \text{ m}^2$ . The dwelling is designed according to a low-energy standard and has massive walls and floor heating. A total of  $33 \text{ m}^2$  of windows with a  $g$ -value of 0.6 is integrated as follows:  $9.5 \text{ m}^2$  on east,  $10.4 \text{ m}^2$  on south and  $11.6 \text{ m}^2$  on west.

Figure 5 shows the training and validation datasets. Both sets consist of one week of hourly data. Although the 'measurement' data comes from a (detailed) simulation, only a few variables that could easily be measured in a real dwelling are used for this case study. These include: the ambient temperature  $T_{Amb}$ , global solar radiation on a horizontal surface  $IGloHor$ , electricity consumption  $powEle$ , thermal power of the heating system  $qHeaCoo$ , and zone temperature  $TZon$ . Hereafter the detailed simulation data is called measurement data. All of the inputs are eliminated from the problem, as discussed in Section 2.5, except for  $TZon$  which is the fitting variable.

Nine models with different order, equations, and parameters have been identified using Latin hypercube sampling. After validation on single-model level, three models have been eliminated because some of their parameters were positioned on a specified minimum or maximum boundary. The RMSE values for auto

and cross-validation of the six remaining models are shown in Figure 6. This figure shows that a lower RMSE on auto-validation does not necessarily imply a lower RMSE on cross-validation. The best model is the one with the lowest RMSE on cross-validation. For this case it is the model with 11 parameters, we'll call it  $model_{11}$ . This is the model for which the structure is shown in Figure 4. The models with 12 and 13 parameters are overfitted.

We will now analyse the results for  $model_{11}$  in more detail. The sample size was 38 and a maximum CPU time of 5.5 seconds was set. The default solver settings were not changed. This implies that the number of collocation elements corresponds to the number of data points ( $n_e = 168$ ), and each collocation element has two collocation points ( $n_c = 2$ ). From the 38 cases, 10 converged to a solution within the limit of 5.5 seconds. Figure 7 shows the RMSE (auto-validation) compared to the IPOPT objective function for each of these 10 solutions. This plot reveals that 4 local optima have been found; the (supposed) global optimum has been found 3 times. The results also show that the discretisation in the collocation method did not lead to significant errors: there is a strong consistency between the optima found according to Equation (3) and the RMSE of the post-simulation.

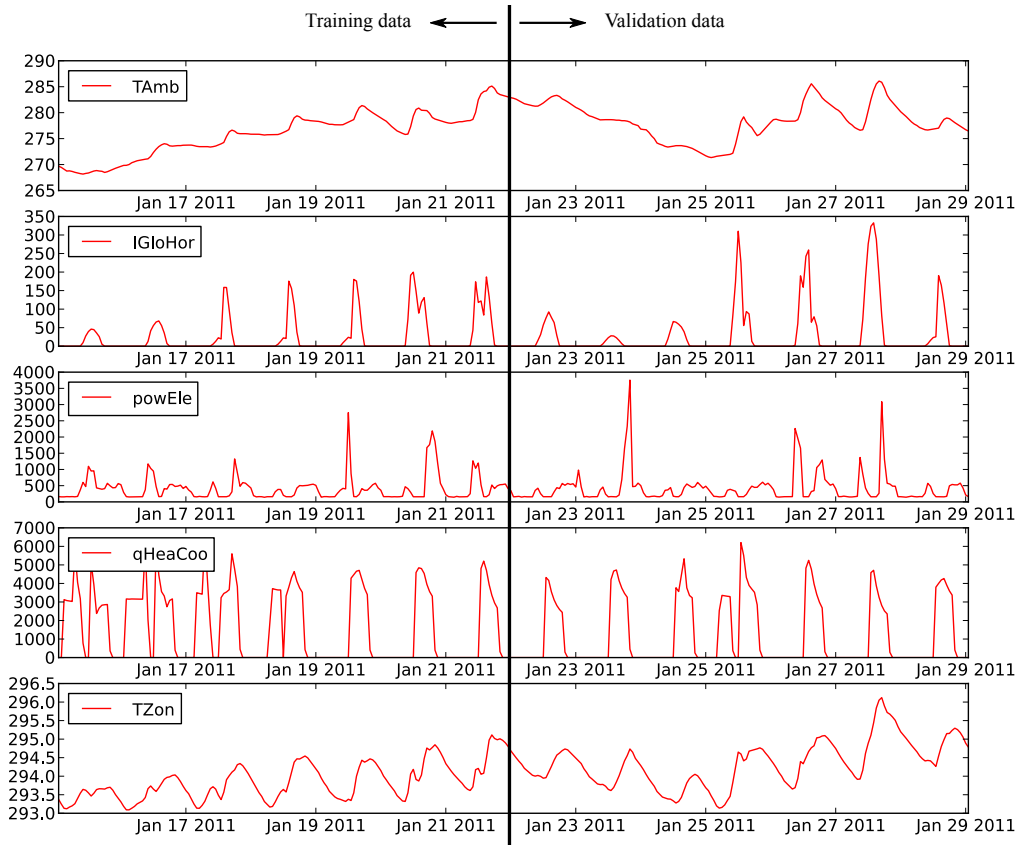


Figure 5: Measurement data for two weeks (first week as training data, second week as validation data)

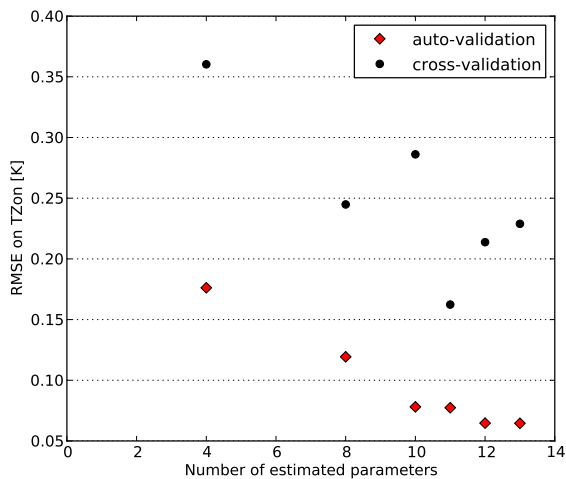


Figure 6: RMSE values for auto and cross-validation for the different models as a function of the number of estimated parameters

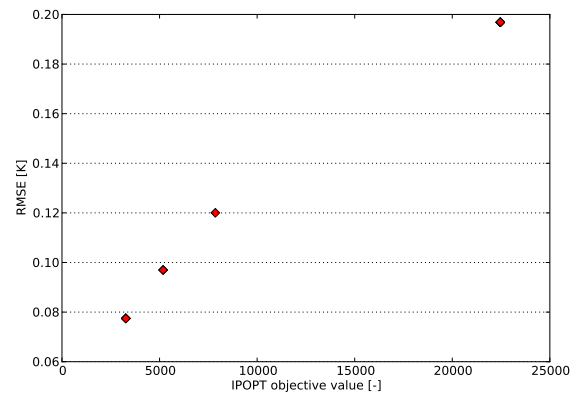


Figure 7: Relation between the IPOPT objective value and RMSE for auto-validation (obtained by post-simulation)



A time series plot of the cross-validation is shown in Figure 8. It is important to note that this plot shows an open-loop simulation over one week. The prediction power of the model is very good, with absolute deviations of the zone temperature always below 0.5 K.

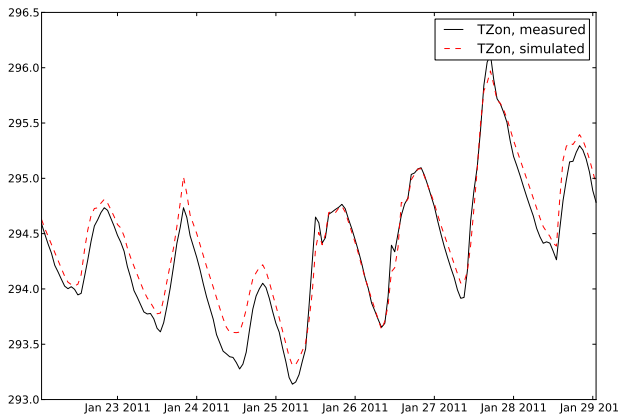


Figure 8: Comparison of the model output with the measurement data for an open loop simulation on the validation dataset

The normalized confidence intervals are shown in Figure 9. All parameters seem to have well defined confidence intervals. The most unconfident parameter values are found for the thermal resistance of the infiltration (resInf in Figure 4) and for the thermal capacity of the zone air (capZon in Figure 4).

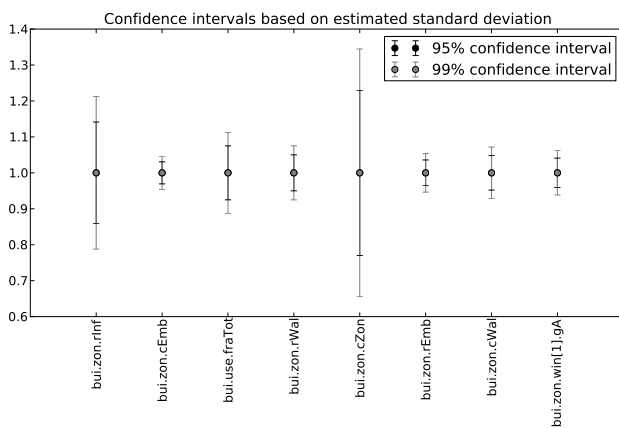


Figure 9: Normalized confidence interval for the parameters in the selected model

## 5 Conclusion

Inverse modelling is gaining attention in the building simulation community. More specifically grey-box modelling is considered as a strong framework for the creation of low-order models for analysis and control of monitored buildings. This paper presents an approach to obtain useful grey-box models in a largely automated way.

The first step is the creation of a building library with many potential model candidates. The Modelica package *FastBuildings* contains low-order models for thermal zones, HVAC, users, single and multi-zone buildings.

Next, a toolbox is presented that largely automates the parameter estimation of the *FastBuildings* models. It is implemented as a Python module that wraps the functionality of *JModelica.org* and presents the user a high-level interface for all common operations. The use of a gradient-based method allows an efficient numerical solution of the estimation problems. Specific attention is paid to robustness and ease-of-use. A Latin hypercube sampling of the parameter search space overcomes issues related to the non-convexity of the optimization problem.

The toolbox is applied to a model order reduction case study for a single-family dwelling. Only variables that can easily be measured in a real building are used. The selected model has 11 parameters and is able to predict the indoor temperature in an open-loop simulation (with a priori knowledge about weather and electricity consumption) with an RMSE of 0.16 K.

The real value of the toolbox can only be assessed by using the obtained models for model predictive control or for large-scale district simulations. Both applications are foreseen in future work.

## Acknowledgement

Roel De Coninck wishes to acknowledge the EU ITEA2 project *Enerficiency* for supporting his work on behalf of 3E and the EU FP7 project *PerformancePlus* (contract nb. 308991) for supporting his work on behalf of KU Leuven. Fredrik Magnusson and Johan Åkesson gratefully acknowledge support from the Lund Center for Control of Complex Engineering Systems (LCCC) and the ELLIIT Excellence Center at Lund University.

## References

- [1] T. Bohlin, “Editorial - Special issue on grey box modelling,” *International journal of adaptive control and signal processing*, vol. 9, pp. 461–464, 1995.
- [2] H. Madsen and J. Holst, “Estimation of continuous-time models for the heat dynamics of a building,” *Energy and Buildings*, vol. 22, pp. 67–79, 1995.
- [3] N. R. Kristensen, H. Madsen, and S. B. Jorgensen, “Parameter estimation in stochastic grey-box models,” *Automatica*, vol. 40, pp. 225–237, Feb. 2004.
- [4] D. B. Crawley, J. W. Hand, M. Kummert, and B. T. Griffith, “Contrasting the capabilities of building energy performance simulation programs,” *Building and Environment*, vol. 43, pp. 661–673, Apr. 2008.
- [5] M. Wetter, “A view on future building system modeling and simulation,” in *Building performance simulation for design and operation* (J. L. M. Hensen and R. Lamberts, eds.), no. i, p. 28, 2011.
- [6] M. Wetter and C. Van Treeck, “IEA EBC Annex 60 - New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards.” <http://iea-annex60.org/about.html>, 2013.
- [7] R. Baetens, R. De Coninck, J. Van Roy, B. Verbruggen, J. Driesen, L. Helsen, and D. Saelens, “Assessing electrical bottlenecks at feeder level for residential net zero-energy buildings by integrated system simulation,” *Applied Energy*, no. (Special issue on Smart Grids, Renewable Energy Integration, and Climate Change Mitigation - Future Electric Energy Systems), 2012.
- [8] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, “Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems,” *Computers and Chemical Engineering*, vol. 34, pp. 1737–1749, Nov. 2010.
- [9] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, *et al.*, “The functional mockup interface for tool independent exchange of simulation models,” in *Modelica’2011 Conference, March*, pp. 20–22, 2011.
- [10] F. Magnusson and J. Åkesson, “Collocation methods for optimization in a Modelica environment,” in *9th International Modelica Conference*, (Munich, Germany), Sept. 2012.
- [11] A. Wächter and L. T. Biegler, “On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [12] HSL, “A collection of Fortran codes for large scale scientific computation.” <http://www.hsl.rl.ac.uk>, 2013.
- [13] J. Andersson, J. Åkesson, and M. Diehl, “CasADi – A symbolic package for automatic differentiation and optimal control,” in *Recent Advances in Algorithmic Differentiation* (S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, eds.), Lecture Notes in Computational Science and Engineering, (Berlin), Springer, 2012.
- [14] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.
- [15] P. Englezos and N. Kalogerakis, *Applied Parameter Estimation for Chemical Engineers*, vol. 81 of *Chemical Industries*. CRC Press, Oct. 2000.
- [16] P. Bacher and H. Madsen, “Identifying suitable models for the heat dynamics of buildings,” *Energy and Buildings*, vol. 43, pp. 1511–1522, Feb. 2011.
- [17] KU Leuven and 3E, “open-IDEAS source code repository.” <https://github.com/open-ideas>, 2014.

# Interfacing Models for Thermal Separation Processes with Fluid Property Data from External Sources

Kai Wellner<sup>\*1</sup>, Carsten Trapp<sup>2</sup>, Gerhard Schmitz<sup>1</sup> and Francesco Casella<sup>3</sup>

<sup>1</sup>Hamburg University of Technology, Institute of Thermo-Fluid Dynamics  
Denickestraße 17, 21075 Hamburg, Germany

<sup>2</sup>Delft University of Technology, Propulsion and Power, Kluyverweg 1, 2629 HS Delft, The Netherlands

<sup>3</sup>Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria  
Via Ponzio 34/5, 20133 Milano, Italy

## Abstract

So far, when modelling processes that demand for multi-phase and multi-component fluid property data, the user has to implement the required media models in the Modelica language as these types of fluids are not supported by Modelica.Media. This paper presents a first approach on how to implement fluid property data in process models of an existing library from external sources and highlights which problems have to be overcome. Furthermore, it provides recommendations for the design of an efficient and user-friendly interface to external media packages.

*Keywords: thermal separation; two-phase; multi-component mixture; external fluid property data; transient simulation*

## 1 Introduction and motivation

Using existing model libraries for the modelling of complex chemical processes is especially challenging because it can be hard to comprehend the structure of the models. Nevertheless, when using these models one might want to implement different media models from what is already available. One convenient way is accessing external property packages instead of developing and implementing required fluid property models in the Modelica language. In this context it might be necessary to provide, next to the primary thermodynamic property data, also property derivatives.

The main objective is to develop an interface between existing Modelica libraries such as the `ThermalSeparation` library and fluid property packages, which successfully deal with situations where

the total time derivative of fluid properties is required. This can be the case for several reasons:

- The formulation of the model equations results in a higher index problem.
- The system states do not equal the variables in the derivative operator.
- There are more variables in the derivative operator than there are differential equations.

The simulation tool handles all three situations the same way which is by applying Pantelides' algorithm and deriving the necessary equations in this process [1]. In contrast to the first bullet point, the last bullet points do not necessarily result in a high index problem. But in both cases additional equations have to be derived by time (which can include fluid property relations) to solve the whole set of equations. In the following this technique is referred to as automated index reduction, regardless of the origin of the problem.

In comparison to developing models from scratch where it is possible to maintain an index-1 form of the system, reducing the index of existing models is in general not feasible. Further information on the differential index and index reduction is provided by Fritzson [2].

An earlier attempt to couple the `ThermalSeparation` library to an external property package still lacked a proper interface [3]. Later on a coupling to external fluid properties was successful although the balance equations had to be modified for this purpose because no derivatives were available and the automated index reduction failed [4].

Some general advantages and reasons for the need to interface external thermodynamic property data

<sup>\*</sup>kai.wellner@tuhh.de, schmitz@tuhh.de

with Modelica models are summarized in the following:

- The Modelica language is increasingly used for process modelling in the field of chemical engineering which entails the modelling of multi-phase, multi-component media. Currently, this type of fluids are not supported by Modelica.Media. In case the user needs such media models, an implementation from scratch is required. This also involves coding of flash calculations in order to determine the equilibrium composition for example for a given  $p$ ,  $T$ ,  $x_{total}$ . Depending on the complexity of the fluid mixture and the used equation of state (EoS), the implementation can be quite cumbersome, time-consuming and error-prone.
- Using existing property packages provides a higher flexibility in terms of available components and EoS. For a Modelica model library developed for a wide range of different applications it is typically not practical or even possible to implement a large number of media models covering any application case. Therefore, providing a user-friendly interface to external property packages is an important feature.
- Calculating property data in an external environment can significantly increase the simulation speed [5]. However this is not a general feature because in some cases it can have the exact opposite effect and decrease the simulation speed.

## 2 The ThermalSeparation library

In process engineering the purification of a product plays an important role. Usually the intermediate products are not of the desired purity and have to be treated in a downstream process. Such processes often involve thermal separation like absorption in gas-liquid separation columns. As the operation of post-treatments like separation can be highly dynamic, computer simulations are indispensable for a better understanding of the process and its optimization targeting for example energy consumption or control strategy design.

The ThermalSeparation library has been developed for the simulation of absorption and rectification processes. The challenge when interfacing external fluid property data is that in the current implementation the balance equations are written in an way that

forces the simulation tool to apply automated symbolic index reduction. If that happens, implementing fluid property data from external sources demands special requirements. Further information on how the formulation of balance equations can affect the use of external fluid property packages and how these problems can be overcome is given in section 3.

### 2.1 Library Structure

Compared to earlier versions of the ThermalSeparation library [4] a new feature that has been implemented in the framework of this paper is the replaceability of the balance equations which easily allows to switch between different formulations of balance equations.

For this work a modified set of balance equations that does not require automated symbolic index reduction is added to illustrate the differences to the original equations in terms of computational performance. A UML class diagram of the library structure is shown in figure 1.

Vapour and liquid medium models that invoke external fluid property calls are added to the respective classes. Moreover, a film model for the external call of the flash calculations is implemented. The classes that are adapted to use external media data are indicated with a bold frame in the UML diagram.

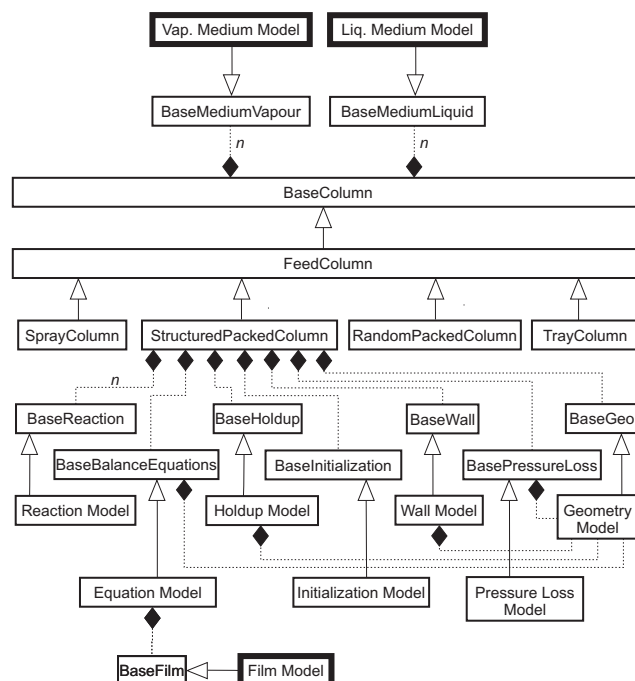


Figure 1: UML diagram of the column model structure.

## 2.2 Interfacing the property package

The use of external sources for the computation of thermodynamic properties is in this paper exemplary demonstrated with the FluidProp package, which is interfaced via the ModelicaFluidProp library [6, 7]. For calculating the actual properties the "PC-SAFT" (perturbed chain statistical associating fluid theory) EoS is employed due to its success in predicting vapour/gas-liquid equilibria of complex fluids and mixtures for a broad range of conditions and due to fact that it also provides the partial derivatives of specific molar enthalpy and specific molar volume with respect to pressure, temperature and composition. These derivatives are necessary to compute the total time derivative (cf. section 3.2). The derivatives are obtained by separate function calls. To enable the use of the total time derivatives in the models of the ThermalSeparation library, the implemented Modelica property functions are modified such that these derivatives are specified by using the derivative annotation. This way the simulation environment can perform automated symbolic index reduction using the specified total time derivatives of the fluid property functions which enables the modeller to perform simulations with variable system states or of higher order index, which would otherwise fail when using external property packages. An example on how the FluidProp calls are implemented in the ThermalSeparation library media models is illustrated in listing 1. The medium definition is programmed in a separate class and mainly consists of string operators that define the fluid constituents and the medium model (EoS) used for the property calculations.

The implemented property functions require as input a string with the thermodynamic state definition (e.g. "PT" for pressure and temperature), the actual values for these states and the overall molar composition. Partial derivatives are currently available with respect to temperature, pressure and composition.

```

model BaseProperties
  ThermalSeparation.Units.MolarEnthalpy h
    "Specific enthalpy of medium";
  ThermalSeparation.Media.Types.Density d
    "Density of medium";
  FluidPropMedium.AllPropsOut prop_mol
    "external property record";
  SI.MoleFraction conc[nSubstance]
    "total composition";
  SI.Pressure p "system pressure";
  SI.Temperature T "system temperature";
equation

```

```

(prop_mol) = Medium.AllPropsMole(
  "PT",
  p,
  T,
  Conc);
h=prop_mol.h;
d=prop_mol.d;
annotation(derivative=
  Medium.AllPropsMole_der);
end BaseProperties;

```

Listing 1: Fluid property call.

The computation of the equilibrium compositions for the vapour-liquid phase boundary is treated in a separate class. The respective function call requires the same inputs as the property calls and returns the equilibrium composition for each phase:

```

model ThermodynamicEquilibrium
  SI.MoleFraction conc[nSubstance]
    "total composition";
  SI.Pressure p "system pressure";
  SI.Temperature T "system temperature";
  SI.MoleFraction x_eq[nSubstance]
    "liq. phase equilibrium composition";
  SI.MoleFraction y_eq[nSubstance]
    "vap. phase equilibrium composition";
equation
  (x_eq, y_eq) =
    Medium.AllProps2phMole(
      "PT",
      p,
      T,
      Conc);
end BaseProperties;

```

Listing 2: Flash calculation call.

## 3 Modelling approach

There are two different approaches when using external fluid property data concerning the modelling structure of the Modelica process models.

In the first approach an automated index reduction is avoided by writing (or rewriting in case of existing models) the balance equations in an explicit manner with respect to the state variables. The external property data can easily be integrated in general without the necessity to provide any partial derivatives.

In the second approach, the balance equations are written in the most "natural" way which might foster numerical robustness and simulation speed or ease of initialization in case of new models. In case of existing models the balance equations are maintained as coded.

Typically this leads to an implicit formulation of the balance equations.

Both approaches are explained in the following:

### 3.1 Explicit modelling of balancing equations

For a separation column with rate-based modelling approach the balance equations for one stage can be written as:

$$\frac{dN_{i,j}^L}{dt} = \dot{N}_{in,j+1}^L \cdot x_{in,i,j+1}^L - \dot{N}_{out,j}^L \cdot x_{out,i,j}^L + \dot{N}_{PB,j}^L \quad (1)$$

$$\frac{dN_{i,j}^V}{dt} = \dot{N}_{in,j-1}^V \cdot x_{in,i,j-1}^V - \dot{N}_{out,j}^V \cdot x_{out,i,j}^V + \dot{N}_{PB,j}^V \quad (2)$$

$$\frac{dU_j^L}{dt} = \dot{N}_{in,j+1}^L \cdot h_{in,j+1}^L - \dot{N}_{out,j}^L \cdot h_{out,j}^L + \dot{Q}_{PB,j}^L \quad (3)$$

$$\frac{dU_j^V}{dt} = \dot{N}_{in,j-1}^V \cdot h_{in,j-1}^V - \dot{N}_{out,j}^V \cdot h_{out,j}^V + \dot{Q}_{PB,j}^V \quad (4)$$

Where  $N_{i,j}$  refers to the molar content of component  $i$  on stage  $j$  for the liquid and vapour phase respectively.  $\dot{N}_{in}$  and  $\dot{N}_{out}$  are the molar flow rates into and out of the stage. Furthermore,  $x$  is the molar fraction,  $h$  is the specific molar enthalpy,  $\dot{N}_{PB}$  and  $\dot{Q}_{PB}$  are the mole and energy flow rates across the phase boundary and are calculated using constitutive equations. Figure 2 shows a schematic diagram of such column stage.

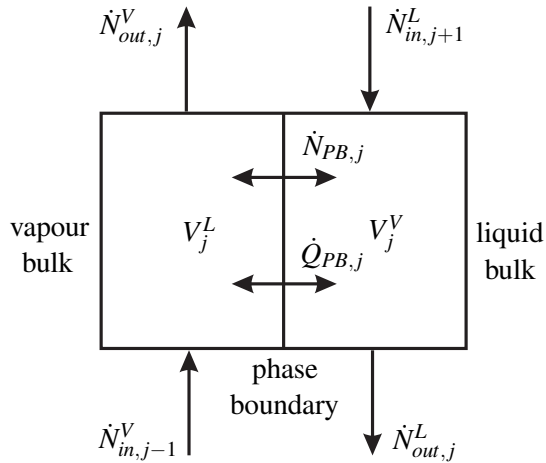


Figure 2: Schematic diagram of a column stage.

In order to avoid the differentiation of additional equations and therefore the necessity to provide total time derivatives of the fluid properties, the system states have to be the same as the differentiated variables, in this case  $N_i^L$ ,  $N_i^V$ ,  $U^L$  and  $U^V$ . To further increase simulation speed, the system states should equal the thermodynamic states, which are the inputs to the property calls. In this way, the fluid properties

can be directly computed from the state variables and therefore no iterations are required. For this purpose the left hand sides of the differential equations have to be rewritten as a function of the thermodynamic state variables. A recommendation how this can be implemented is given in section 5.

Now the model is designed work without fluid property derivatives. However, as soon as the user chooses a different state variable as the ones named above, additional derivatives are required again. Hence, these types of models are not flexible because variable states can be an advantage in model flexibility as they provide additional robustness.

### 3.2 Implicit modelling of balance equations

Using existing model libraries with external fluid properties can be somewhat more complicated because the balance equations may not always be modelled in an explicit manner as explained in equations (1)-(4). For example in the ThermalSeparation library the balance equations are formulated like this:

$$\frac{d(V_j^L \cdot c_{i,j}^L)}{dt} = \dot{N}_{in,j+1}^L \cdot x_{in,i,j+1}^L - \dot{N}_{out,j}^L \cdot x_{out,i,j}^L + \dot{N}_{PB,j}^L \quad (5)$$

$$\frac{d(V_j^V \cdot c_{i,j}^V)}{dt} = \dot{N}_{in,j-1}^V \cdot x_{in,i,j-1}^V - \dot{N}_{out,j}^V \cdot x_{out,i,j}^V + \dot{N}_{PB,j}^V \quad (6)$$

$$\frac{d[V_j^L \cdot \sum_{i=1}^n (u_j^L \cdot c_{i,j}^L)]}{dt} = \dot{N}_{in,j+1}^L \cdot h_{in,j+1}^L - \dot{N}_{out,j}^L \cdot h_{out,j}^L + \dot{Q}_{PB,j}^L \quad (7)$$

$$\frac{d[V_j^V \cdot \sum_{i=1}^n (u_j^V \cdot c_{i,j}^V)]}{dt} = \dot{N}_{in,j-1}^V \cdot h_{in,j-1}^V - \dot{N}_{out,j}^V \cdot h_{out,j}^V + \dot{Q}_{PB,j}^V \quad (8)$$

Where  $V$  is the volume,  $c$  is the concentration and  $u$  is the specific internal energy.

Experience has shown that this way of modelling balance equations in the ThermalSeparation library provides the best performance in terms of numerical robustness and robust initialization. Moreover, to provide the full flexibility it is necessary to be able to choose the system states freely. Further information on this is given in Dietl [4].

It is obvious that the number of differentiated variables ( $V_j^L$ ,  $V_j^V$ ,  $c_{i,j}^L$ ,  $c_{i,j}^V$ ,  $u_{i,j}^L$  and  $u_{i,j}^V$ ) is higher than the number of differential equations (5)-(8). This results in a system that can only be solved if the missing

derivatives are provided by deriving additional equations. In the present case this also includes the differentiation of fluid properties, which subsequently causes the compilation to fail as the external property functions cannot be differentiated by the simulation environment. One way to circumvent the compilation failure without changing the governing equations is to provide the total time derivatives for the property data according to equations (9) and (10). Therefore, the partial derivatives of the properties with respect to the system states are required. Equations (9) and (10) are contained in the function that is invoked by the derivative annotation in listing 1. In the column model the specific molar volume  $v$  is used to convert the molar fraction  $x$  to the concentration  $c$ .

$$\frac{dh}{dt} = \left(\frac{dh}{dp}\right)_{T,x_i} \cdot \left(\frac{dp}{dt}\right) + \left(\frac{dh}{dT}\right)_{p,x_i} \cdot \left(\frac{dT}{dt}\right) \quad (9)$$

$$+ \left(\frac{dh}{dx_i}\right)_{p,T} \cdot \left(\frac{dx_i}{dt}\right)$$

$$\frac{dv}{dt} = \left(\frac{dv}{dp}\right)_{T,x_i} \cdot \left(\frac{dp}{dt}\right) + \left(\frac{dv}{dT}\right)_{p,x_i} \cdot \left(\frac{dT}{dt}\right) \quad (10)$$

$$+ \left(\frac{dv}{dx_i}\right)_{p,T} \cdot \left(\frac{dx_i}{dt}\right)$$

In contrast to this a higher index problem in chemical process models can arise for example because of the volume constraint when assuming incompressibility of the fluid or when chemical reactions are modelled in an equilibrium fashion [8].

## 4 Example of use

In the following the implementation of both approaches (use of explicit balance equations and use of implicit balance equations) is demonstrated and compared for the simulation of a rate-based column model employing a fluid mixture of n-propane and n-pentane using external property data. Both approaches of modelling balance equations are successfully implemented, which has been proven by the fact that with both models the same solution has been obtained.

Inside the medium models the specific volume, the specific enthalpy and the specific heat capacity are used from FluidProp for both vapour and liquid phase. The specific internal energy can be directly calculated from the specific enthalpy. Additionally, the equilibrium composition is used in the film model. The property calls are invoked using pressure and temperature

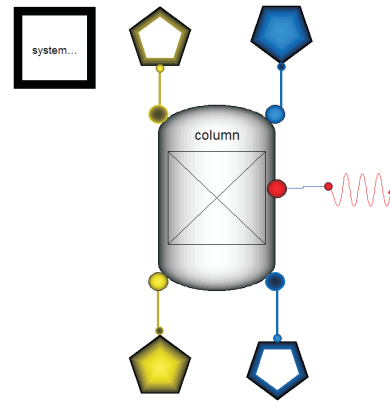


Figure 3: Diagram of the Modelica column model.

as thermodynamic states. The system states of the column model are pressure, temperature and molar composition of liquid and vapour phase.

Figure 4 depicts the comparison in simulation time of both models, implicit and explicit balance equations, for a step change in the feed flow rate at 500 seconds when the system has reached a steady-state after initialization.

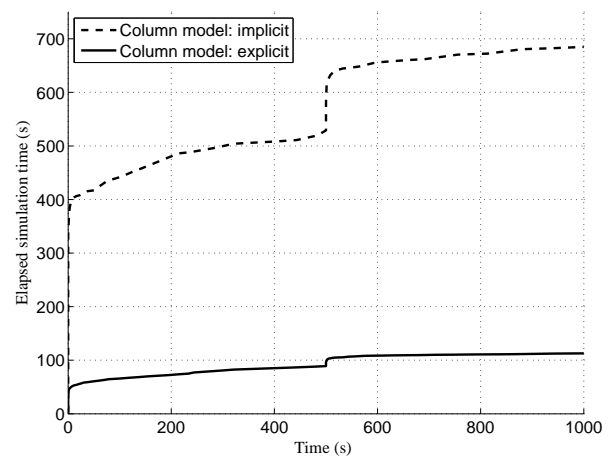


Figure 4: Comparison of simulation times.

When comparing both models it can be observed that the simulation time for the implicit model is much higher. This is explained by the fact that the calculation of partial derivatives is computationally very expensive which can be seen in sections with high gradients like initialization or change of boundary conditions. There the simulation of the implicitly modelled system slows down significantly in comparison to the model that does not use external derivatives. Thus it can be concluded that from a performance point of view it is advantageous to model the system in an explicit manner in order to avoid the necessity to com-

pute partial derivatives.

While the calculation of the external fluid properties requires the biggest part of the computational effort (especially when partial derivatives are needed), it also possesses a large room for improvement and thus for the decrease of the overall simulation time.

Aside from the external calculation of the partial derivatives there are other possible sources for the slowdown of the simulation in this specific case. First of all this modelling approach combined with the use of external property data introduces numerical jacobians that have to be calculated in order to obtain the solution of the system. These jacobians can both destabilize and slow down the simulation. Furthermore, implicit modelling of the balancing equations in combination with the use of external derivatives can lead to larger blocks of non-linear system of equations that have to be solved iteratively as tearing cannot be applied in an efficient way. Table 1 shows a comparison for the sizes of non-linear system of equations between explicit and implicit modelling approaches for the example of use.

	Implicit model	Explicit model
(1)	{58, 52, 68, 1, 1, 1, 1}	{58, 50, 54, 1, 1}
(2)	{6, 2, 56, 0, 0, 0, 0}	{8, 2, 6, 0, 0}

Table 1: Sizes of non-linear system of equations before (1) and after manipulation (2).

The presented simulations are using rate-based column models from the `ThermalSeparation` library, which means that in terms of external property computations one call is required for each phase and one additional call for the film model in order to determine the thermodynamic equilibrium. If the partial derivatives are needed they are invoked via the derivative annotation additionally for each property call. Having three property calls per column stage is obviously computationally expensive, especially when derivatives are calculated, in comparison to e.g. an equilibrium stage model where only one property call per column stage would be sufficient.

## 5 Recommendations for improvement of the interface and balance equation modelling

The current interface does not provide the possibility to choose between function calls with or without the

output of the total time derivatives via the derivative annotation. If they are needed, they have to be calculated from partial derivatives which have to be invoked with a separate function call. For a future interface it should be possible to opt between functions that provide derivatives and such that do not. In principle it would be sufficient to have one function and the derivatives in the annotation are only calculated when needed. In practice however this does not work because for complex models (like in the example of use) the derivatives are also computed when they are not needed. So on the one hand calculating the derivatives all the time consumes too much CPU time if it is not required but on the other hand sometimes the derivatives are absolutely necessary.

Furthermore it would be convenient if the user can specify the needed derivatives to be able to adapt them to the system states of the model that the derivatives are used in (e.g. the column model). Listing 3 shows a suggestion how such an interface can look like for both options, with and without output of derivatives. The design is inspired by the current implementation and expanded with a `NumericalState` record in case the derivatives are required. The derivatives in the `prop_der.dh[:]` array are arranged in the same order as the numerical states inside the `NumericalState` record. Although these recommendations are related to the current interface of the `ModelicaFluidProp` library they can be taken as general recommendations for the design of future interfaces.

```
(prop) = Medium.AllProps(
    ThermodynamicState,
    Value1,
    Value2,
    Conc);
h=prop.h;

(prop_der) = Medium.AllPropsDer(
    ThermodynamicState,
    Value1,
    Value2,
    Conc,
    NumericalState);
h=prop_der.h;
dhdp=prop_der.dh[1];
dhdT=prop_der.dh[2];
dhdx[1]=prop_der.dh[3];
dhdx[2]=prop_der.dh[4];
```

Listing 3: Design recommendation for fluid property interface

In addition, the general way of modelling balance equations for processes involving multi-phase and multi-component fluids should be reconsidered. The



modelling of balance equations with mass, internal energy and mass fractions originates from the modelling of energy systems. This approach might not be optimal for problems in the field of chemical engineering. For chemical processes the balance equations should rather be set up with volume related properties like for example density or the molar specific volume as explained in Pantelides [8]. For an equilibrium stage model the left hand side of the mole balance can be rewritten according to equation (11).

$$\frac{dN_i}{dt} = V \cdot \frac{d(v \cdot x_i)}{dt} \quad (11)$$

The advantage of this structure is that fluid property calls can directly be invoked with the system states "u, $\rho$ " or "u,v" (depending on whether the model is on mass or molar basis) and still an automated index reduction due to the need of additional derivatives is avoided.

## 6 Summary and conclusions

This paper demonstrates and compares two approaches on the use of external property packages for the computation of fluid data interfaced to Modelica process models. With the exemplary column model using an explicit formulation of the balance equations the use of partial derivatives is avoided. This model is in terms of computational speed favourable in comparison to the model with an implicit formulation of the balance equations. However, especially when utilizing existing model libraries rewriting balance equations is not convenient or feasible. The paper shows that in this case the external property package has to provide partial derivatives of the fluid properties with respect to the system states. This enables the simulation environment to perform automated symbolic index reduction and maintains the flexibility of choosing the state variables freely.

## 7 Acknowledgements

This research project is supported by the Federal Ministry of Economics and Technology (project number 03ET2009).



## Nomenclature

$\dot{N}$	molar flow rate (mol/s)
$\dot{Q}$	heat flow rate (W)
$c$	concentration (mol/m <sup>3</sup> )
$h$	specific molar enthalpy (J/mol)
$N$	amount of substance (mol)
$p$	pressure (Pa)
$T$	temperature (K)
$U$	internal energy (J)
$u$	specific internal energy (J/mol)
$V$	volume (m <sup>3</sup> )
$v$	specific molar volume (mol/m <sup>3</sup> )
$x$	mole fraction (mol/mol <sub>tot</sub> )

## Superscripts

$L$	liquid phase
$V$	vapour phase

## Subscripts

$i$	component $i$
$in$	inflow stream/attribute
$j$	column stage $j$
$out$	outflow stream/attribute
$PB$	phase boundary

## References

- [1] C. Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM journal of scientific computing*, 9:213 – 231, 1988.
- [2] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. ISBN 0-471-47163-1. Wiley, June 2000.
- [3] A. Joos, K. Dietl, and G. Schmitz. Thermal Separation: An Approach for a Modelica Library for Absorption,

- Adsorption and Rectification. In *Proceedings 7th Modelica Conference, Como, Italy, September 20-22, 2009*.
- [4] K. Dietl. *Equation-Based Object-Oriented Modelling of Dynamic Absorption and Rectification Processes*. Dissertation, Hamburg University of Technology, Institute of Thermo-Fluid Dynamics, Hamburg, 2012. ISBN 978-3-8439-0778-1.
- [5] J. Brunnemann, F. Gottelt, K. Wellner, A. Renz, A. Thüring, V. Roeder, C. Hasenbein, C. Schulze, G. Schmitz, and J. Eiden. Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO<sub>2</sub> Capture. In *Proceedings 9th Modelica Conference, Munich, Germany, September 3-5, 2012*.
- [6] P. Colonna, T. van der Stelt, and A. Guardone. FluidProp: a program for the estimation of thermophysical properties of fluids. Version 2.4, software (2004-2012). <http://www.FluidProp.com>.
- [7] C. Trapp, F. Casella, and P. Colonna. Use of External Fluid Property Code in Modelica for Modelling of a Pre-combustion CO<sub>2</sub> Capture Process Involving Multi-Component, Two-Phase Fluids. Submitted to the 10th Modelica Conference, Lund, Sweden, March 10-12, 2014.
- [8] C. Pantelides. The Mathematical Modelling of the Dynamic Behaviour of Process System, October 2000.

# Development of a Real-Time Fuel Processor Model for HIL Simulation

Karin Fröjd<sup>1</sup>, Karin Axelsson<sup>2</sup>, Ivar Torstensson<sup>1</sup>, Erik Åberg<sup>1</sup>, Erik Osvaldsson<sup>2</sup>, Gregor Dolanc<sup>3</sup>, Bostjan Pregelj<sup>3</sup>, Jonas Eborn<sup>1</sup>, Jens Pålsson<sup>1</sup>

<sup>1</sup> Modelon AB, Ideon Science Park, Scheelev. 17, 223 70 Lund, Sweden

<sup>2</sup> PowerCell AB, Ruskvädersgatan 12, 418 34 Göteborg, Sweden

<sup>3</sup>J. Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia

[Karin.Frojd@modelon.com](mailto:Karin.Frojd@modelon.com), [Karin.Axelsson@powercell.se](mailto:Karin.Axelsson@powercell.se)

## Abstract

In this article a real-time model for dynamic simulation of a fuel processor is presented. The model is intended for HIL testing of the PLC for a truck Auxiliary Power Unit (APU) system.

The APU comprises a PEM fuel cell and fuel processor to enable direct utilization of on-board diesel. The system is under development in FCGEN, an EU project under the FP7 program FCH JU [1]. One critical challenge is to design the control system (PLC) to ensure failsafe and environmental friendly startup and operation. The startup phase of the fuel processor is the most critical part, since it is a highly dynamic process involving several complex reactors. It is advantageous to verify the control system before the fuel processor is assembled to avoid possible breakage of components. Such verification can be done with a real-time model representing the physical system. In this study such a model is created using Modelica and Dymola. It is shown that it is possible to load and execute a real-time Modelica model capable of realistically mimicking the system response on a HIL platform. The model runs in real time using a first order explicit (Euler) solver with a time step size of 25 ms.

*Keywords: HIL simulation; fuel reformation; fuel processor, fuel cell, PLC, real-time simulation*

## Abbreviations

APU	Auxiliary Power Unit
ATR	Auto-Thermal Reformer
BoP	Balance of Plant
CAB	Catalytic After Burner
CSB	Catalytic Start Burner

DS	Desulphurizer
FPM	Fuel Processing Module
HIL	Hardware In the Loop
PEM	Proton Exchange Membrane
PLC	Programmable Logic Controller
PrOx	Preferential Oxidation
WGS	Water Gas Shift

## 1 Introduction

Fuel cell systems are an attractive technology for Auxiliary Power Units for e.g. trucks, because of the high efficiency and low emissions. However, PEM fuel cells operate most efficiently with hydrogen, while at gas stations usually only liquid fuels like diesel and gasoline are available. To circumvent this problem a fuel processor may be used to convert the high order hydrocarbons to a hydrogen rich gas mixture. The aim of the FCGEN [1] program is to design and demonstrate such an APU.

The fuel processor is a complicated system of reactors, heat exchangers and BoP components. System startup needs to be carefully designed to avoid poisoning of reactors and system failure and ensure as short startup time as possible. The control system must be designed to give a smooth and secure startup. Verification and adjustment of the control system is normally performed against the real system. However since the system is complex, such testing may become very expensive and time consuming. Hence it is advantageous to test the system against a model in a HIL (Hardware-In-the-Loop) setup before assembling the full system. In this work such a model is developed and tested.

An example of a previous related work is a physical model made in Modelica of a food processor line tested in a HIL set up in 2008 [2].

## 2 Fuel processor

The fuel processor module (FPM) is based on an Auto-Thermal Reformer (ATR) which converts diesel to syngas through steam reformation. Required heat for the endothermic reformation reaction is supplied by oxidation with a limited amount of air.

A desulphurizer (DS) is added downstream of the ATR to remove sulphur to protect downstream catalysts and the fuel cell.

The PEM fuel cell tolerates only very low CO concentrations (ppm levels). To remove CO from the syngas to the fuel cell, a water-gas shift (WGS) reactor and a preferential oxidizer (PrOx) are added downstream of the DS. To avoid poisoning of the fuel cell during the startup phase, a bypass route is used. A catalytic afterburner (CAB) is used to clean the exhaust gases before release to the atmosphere. In the early start phase a start system is in operation burning diesel in a catalytic start burner (CSB) to pre-heat the FPM. The system scheme is shown in Fig. 1 below.

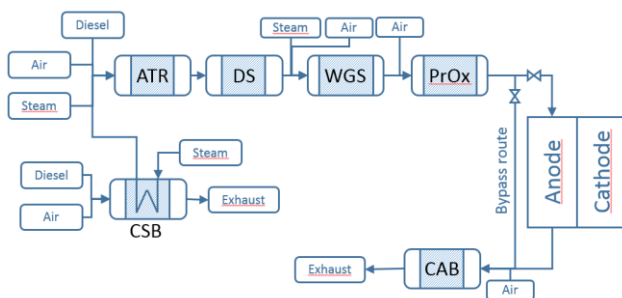


Figure 1: Fuel processor system scheme

The fuel processing module (FPM) is developed by the following partners of FCGEN:

- Volvo Group Trucks Technology, Sweden – coordinator
- PowerCell Sweden AB - plant components, system specification, integration and testing
- Forschungszentrum Jülich GmbH, Germany - ATR and CAB reactors
- Institut für Mikrotechnik Mainz GmbH, Germany - DS, WGS and PrOx reactors
- Johnson Matthey PLC, UK – catalysts for reactors
- Jozef Stefan Institute, Slovenia - control system
- Modelon AB, Sweden – dynamic system model

### 2.1 Startup strategy

The main steps of the startup strategy are;

1. Use the start burner to heat the ATR and downstream reactors to sufficient temperature.
2. Use the start burner to produce steam for the ATR.
3. Ignite the ATR, keep start burner in operation to ensure sufficient fuel and emission conversion.
4. Shut down start burner, start normal operation.
5. Stop by-pass of fuel cell.

## 3 Requirements on HIL model

The aim of this work is to create a model that can be used to test the PLC logic. The simplifications needed to meet the real-time requirement sets a limit on model accuracy. Hence the model will not be suitable nor be used for calibration of PLC logic parameters. The requirements of the HIL simulation model are defined as:

1. The model should be robust and never crash.
2. The PLC requires real-time communication with the plant model. Consequently, no solver step may require a CPU time longer than real time (so called overrun).
3. The model should respond realistically to changes in input signals. For example ATR temperature should respond in the right direction when changing air-fuel ratio or steam mass flow.
4. Trends need to be captured for temperatures and mass flows. Exact numbers are not necessary.
5. Response times are allowed to deviate slightly from reality. In general model response times are longer than real response times, because of the constraint on minimum time constants.

To meet requirement no 2 the model must be fairly simple, and an explicit solver such as the first order Euler method need to be applied. Fast dynamic time scales need to be removed by changing dynamic equations to static. Events need to be eliminated to avoid solver reinitialisation and subsequent overruns. At the same time the model needs to respond realistically to signal changes, as defined in requirement no 3. However, because of the simplicity of the model it cannot be expected to correctly predict absolute temperatures

or times of the system. This accuracy is not needed for the PLC tests.

## 4 Model description

The Modelica model is based on the Fuel Cell Library from Modelon [3]. This newly released product contains a number of component models and examples aimed at PEMFC and other fuel cell applications including fuel processing reactors. The library depends on Modelon Base Library for base classes common to several Modelon libraries. Early work and cornerstone of FCL was carried out by Andersson and Åberg [4]. Dymola features such as drag and drop functionality, graphical user interface, equations in text layer and numerical solvers facilitate development of a dynamic fuel cell model.

The full system includes more than 25 components, including compressors, heat exchangers, valves, reactors and pumps. All relevant components of the fuel processor have been included in the model. The real-time model comprises 217 continuous time states and is of index 1.

Two-phase water medium has been used where necessary. For model robustness pure steam medium was applied wherever possible. Multi-component gas phase media was used for reformat gas and air.

To enable HIL simulations and meet real-time requirement, all extended component models (submodels) need to be simple.

The Dymola model is initially developed for the DASSL implicit ODE solver. To avoid overruns in the HIL simulations an explicit fixed-step solver must be used. The first order Euler solver is applied in this work. To match the real time requirement a sufficiently large time step size need to be used in the Euler solver.

In the following sections the main submodels, real-time adaptations and HIL setup are described.

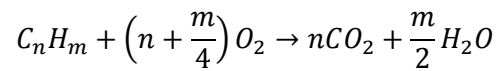
### 4.1 Chemical reactors

Chemistry is very complex and the final composition is highly dependent on the initial state and local phenomena. Simplified chemistry models are usually valid in a narrow state space. The purpose of this study is to validate and test the FPM control system during normal system startup. It is enough to predict reasonable heat release in each reactor. Detailed species concentrations are not required. Hence simple chemistry models designed for the normal startup and operation range are suitable.

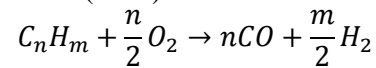
In the ATR fuel is reformed by steam under presence of a limited amount of oxygen. Fuel lean combustion does not need to be covered by the model, and complete conversion is a valid assumption.

The reactors are implemented as homogeneous (lumped) reactor models, utilizing different reaction objects to simulate reaction time characteristics typical for each reactor. The reaction objects used are equilibrium chemistry, complete conversion and reaction kinetics using Arrhenius equation. Heat exchangers, flow losses and heat losses are included where applicable. See Fig. 2 for Dymola model schematics of the WGS reactor. The following chemical reactions are applied in the system:

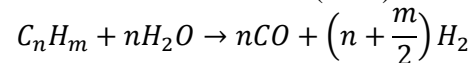
Combustion (ATR):



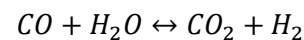
Partial oxidation (ATR):



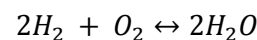
Hydrocarbon steam reformation (ATR):



Water-gas shift (ATR, WGS):



H<sub>2</sub> oxidation (PrOx, CAB):



CO oxidation (PrOx, CAB):

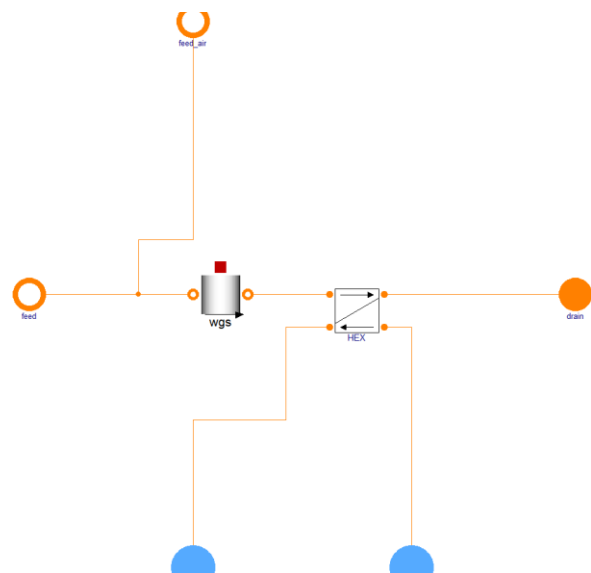
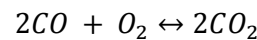


Figure 2: WGS model in Dymola

### 4.2 Start system

The catalytic start burner from Zemission is simply modeled by tabulated temperatures and mass flows as

function of power load. The table based start burner is connected to heat exchangers transferring the heat to the rest of the fuel processor (see Fig. 3). The startup logics involves electrical pre-heating, ignition, certain delays, ramping to operating point etc.

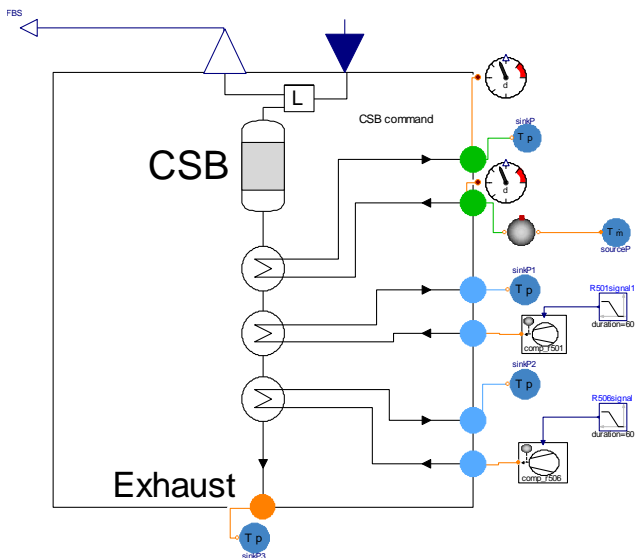


Figure 3: Start system in Dymola

### 4.3 Achieving real-time

A number of model simplifications and changes were introduced to enable real-time simulations:

- Non-linear equations: The submodels were simplified to avoid non-linear equation systems.
- Model simplifications: The model complexity was decreased. The discretized heat exchangers were replaced by lumped models. The discretized ATR, WGS, PROX and CAB reactors were described by lumped reaction volumes connected to lumped heat exchangers.
- Event elimination: The Modelica operator noEvent() was added for applicable functions. Functions not supporting noEvent were exchanged by functions supporting or including noEvent(). In particular IF constructs were removed.
- Minimum flow values: In the real system zero flow or back flow may occur. This can lead to division by zero or very fast dynamics. To avoid this, a negligibly small mass flow is initiated where the real system has zero flow.
- Chemical time scale: The time scale of chemical reactions is generally several orders of magnitude smaller than the required model time step size. Hence the complex dynamic

reaction system is replaced by static equations for most reactors. For reactors with dynamic time scales close to the model time step size the chemical reaction rates are damped sufficiently.

- Time scale for flow: Pressure waves travel fast and yield short time scales. To increase the time scales flow losses and volumes were lumped and increased.
- Simplified media: The higher order original NASA Glenn correlations [5] for thermodynamic properties were replaced by linear correlations for the full temperature range. Hence the possible discontinuity by the break temperature is removed.

To enable offline testing of the fuel processor a simple control system was implemented in Dymola and connected to the bus signals in the fuel processor model. An overview of the control system is shown in Fig. 4. Ramps and tables were used for opening and closing of valves and changing set points for mass flow regulators. Simple integrator controllers were used. The purpose of the model control system is only to test the robustness of the model and the model response; hence it is kept simple.

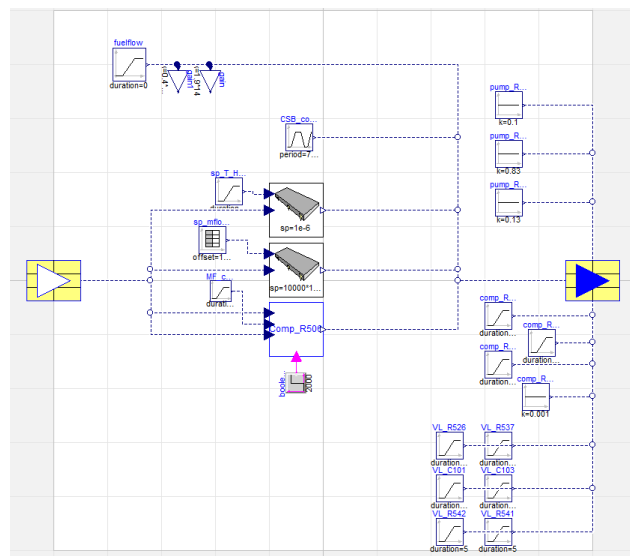


Figure 4: Model control system

### 4.4 HIL setup

The HIL set-up comprises 5 units. The HIL core form the Host computer, where the model is put together and compiled and the Real-time computer, which actually runs the model simulation in real time. The Real-time computer is connected to the PLC running the FCGEN APU control system via CAN buses. For

operation monitoring and adjustment of control system parameters another computer is used running the SCADA-HMI. This computer can be accessed remotely from J. Stefan Institute, allowing frequent checks and test execution without additional costs. The scheme of the HIL set-up is presented in Fig. 5.

The Real-time computer is a Speedgoat performance real-time target machine running xPC Target. The Dymola Simulink interface is used to import and build the Modelica model in Simulink.

The communication to the PLC includes more than 100 sensor and actuator signals. All signals are added to the model and connected in Simulink. An overview of the model setup in Simulink is seen in Fig. 6.

The time step of the model is 25 ms, which is a trade-off between model accuracy and the real time requirement. The sample time is 500 ms to mimic the real system response times. The communication interval to the PLC is 1 ms to ensure sufficient signal transfer rate.

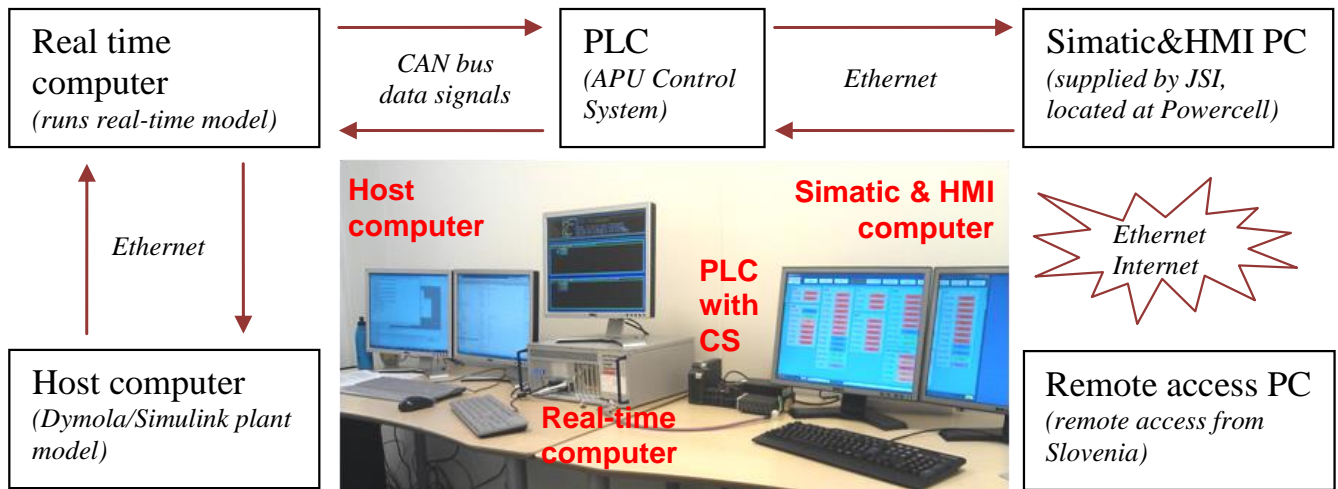


Figure 5: HIL testing assembly scheme

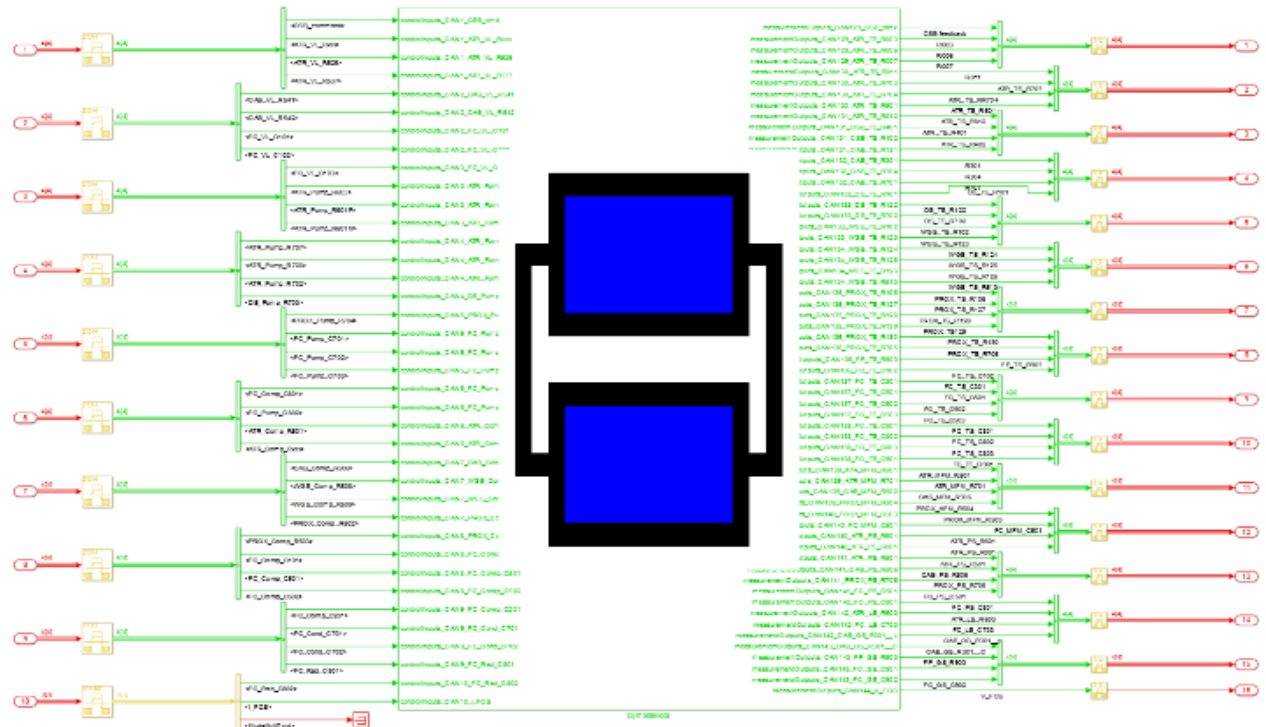


Figure 6: Simulink implementation using the Dymola Simulink interface

### 5 Results and Discussion

The model is tested offline for phase 1 to 4 in the startup strategy (see section 2.1). Air-fuel ratio and steam mass flow is changed to verify the model response. Following the discussion in section 3, the model cannot be expected to deliver correct absolute values or times. Hence temperatures and times are normalized, and absolute values are not presented.

Temperatures through the system are shown in Fig. 7 and 8. Fig. 7 shows the inlet temperatures to the ATR, Fig. 8 shows the reactor exhaust temperatures through the system. The startup events are indicated on the x axis.

The overall dynamics of the model is reasonable; inlet air and steam are heated by the startburner, and the temperature drops when the startburner is turned off. The ATR temperature is increased by startburner heating. By ignition, the temperature is increased, and when CSB is turned off the temperature drops to a stable value.

The transient overshoots seen in the figures arise from the use of simple integrator controllers.

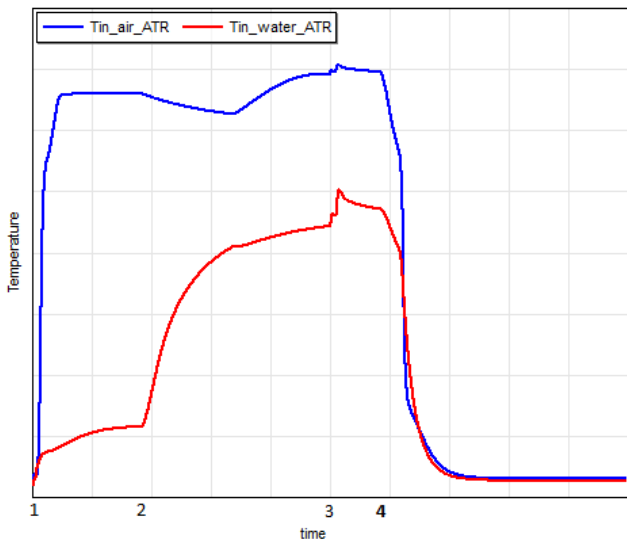


Figure 7: ATR inlet temperatures. The numbers below the x axis indicate start of phase 1-4 defined in section 2.1.

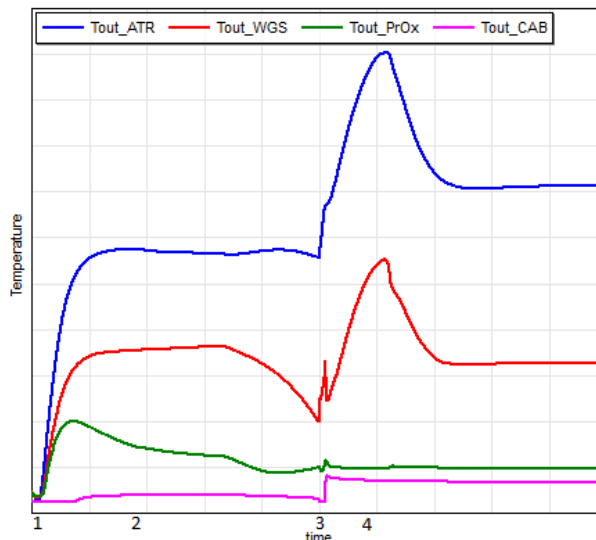


Figure 8: Reactor exhaust temperature. The numbers below the x axis indicate start of phase 1-4 defined in section 2.1.

To test the model response the air-fuel ratio was decreased by lowering the inlet air flow. Since the ATR is run under fuel rich conditions this should lead to decreased ATR outlet temperature. This is confirmed by Fig. 9, where ATR exhaust temperatures for different conditions are plotted.

From this figure it is also confirmed that the ATR temperature is decreased when increasing steam mass flow, as expected.

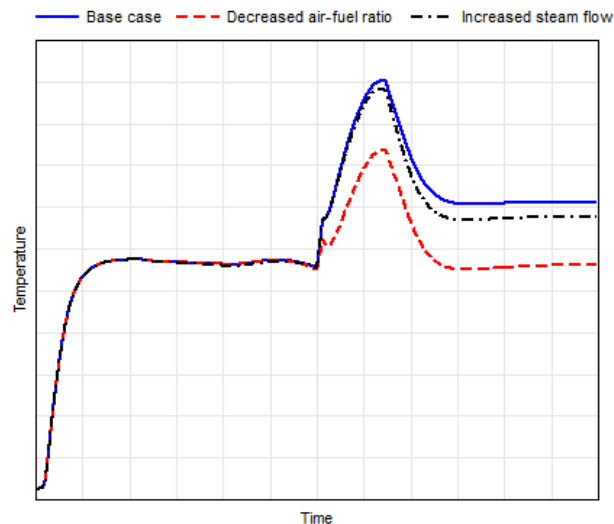


Figure 9: ATR exhaust temperature for different conditions

To meet the real time requirement reactor volumes and flow losses were increased. To evaluate the effect of these necessary changes comparisons of results with and without these changes are shown in Fig. 10 and 11 below. Both simulations were run with the



Dassl solver. The deviation between original and real time model is considered acceptable.

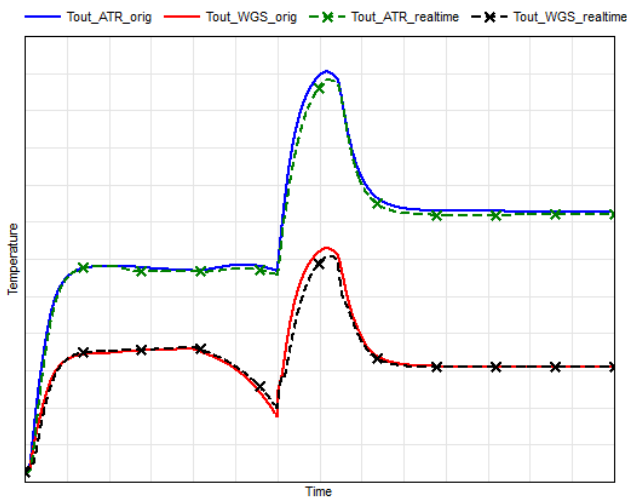


Figure 10: Temperatures before (\*\_orig) and after (\*\_realtime) modifications of volumes and flow losses to eliminate time constants below 25 ms.

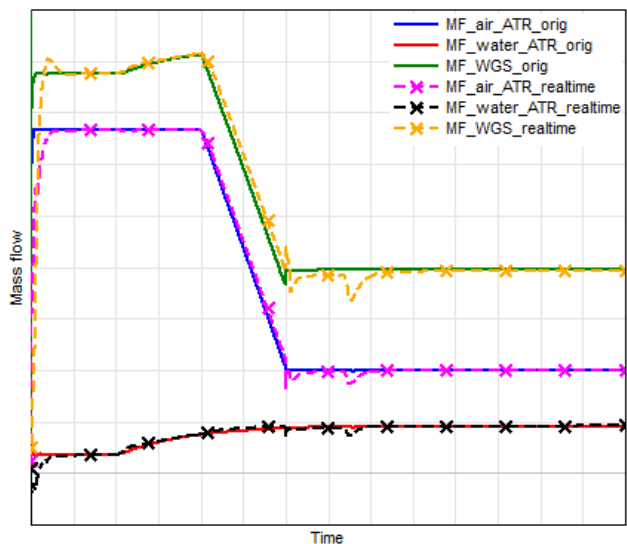


Figure 11: Mass flows before (\*\_orig) and after (\*\_realtime) modifications of volumes and flow losses to eliminate time constants below 25 ms.

Using DASSL on a laptop with an Intel Core™ i7-3740QM CPU @ 2.70 GHz, the model is 68 times faster than real-time. With the Euler solver each time step takes about 4.5 ms without overruns; the model is 5.6 times faster than realtime when using a time step size of 25 ms. The results are confirmed to be consistent using DASSL and Euler, see Fig. 12. At the Speedgoat machine each time step requires slightly more than 20 ms. For this reason a 25 ms step size is chosen for the HIL simulations.

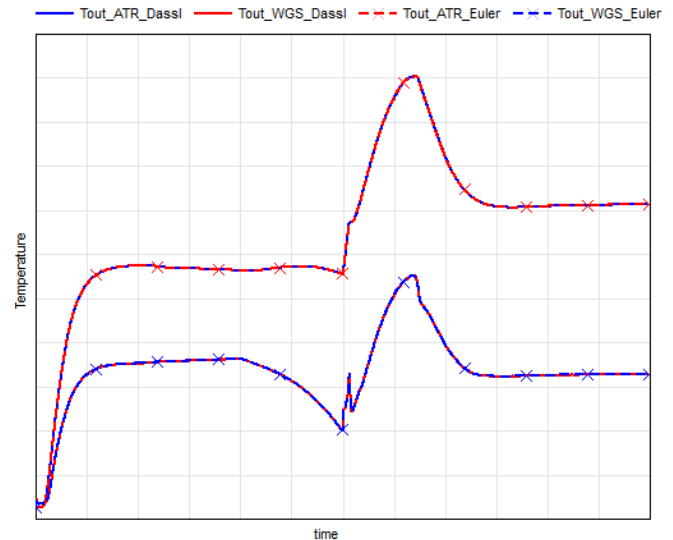


Figure 12: Simulations with DASSL and Euler Solver

The major difficulty in achieving real-time capability was to ensure large enough time constants under all conditions. Lumping volumes in reactors and heat exchangers was not enough. The smallest time constants of the lumped system were  $< 0.05$  ms. To increase the time constants the volumes were increased until the results started to deviate from the original results. Since the time constants were still too small, it was decided to also increase flow losses through the system. This increased the time constants sufficiently while still keeping sufficient accuracy in temperature and mass flows (see Fig. 10 and 11).

The second major difficulty was to ensure stable operation under all circumstances. This was handled by extensive testing of the model, eliminating issues one by one. The major difficulties were connected to back flow and negative mass fractions. Backflow is difficult to consistently handle under all circumstances without event generation and without obtaining non-linear equation systems. Hence a large enough minimum flow from the compressors, pumps and valves was imposed. Negative mass fractions occur when the solver takes a too large step size for a dynamic reaction where one of the reactants is completely consumed. Hence reactions were limited and max limiters were applied on species mass fractions.

The model is currently tested on the SpeedGoat machine using xPC Target. It is confirmed to successfully load and start. Full HIL tests with a PLC will be the subject of future work. Additional model revisions may be required during the course of the PLC tests.

## 6 Conclusions

A real-time model for dynamic simulation of a fuel processor is developed. The following main conclusions are made:

- It is possible to create a real-time capable model using Modelica and Dymola for such a fuel processor system.
- A system of 217 dynamical states needs Euler time steps of about 25 ms to enable HIL simulation on the current hardware configuration.
- Several simplifications were needed to increase the time constants to 25 ms.

## 7 Acknowledgements

This project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) for the Fuel Cells and Hydrogen Joint Technology Initiative under grant agreement n° [277844].

## References

- [1] FCGEN is financially supported by the European Union's Seventh Framework Programme (FP7/2007-2013) for the Fuel Cells and Hydrogen Joint Technology Initiative *under grant agreement n° [277844]*
- [2] Gäfvert, M., Tummescheit, H., Wikander, H., Skoglund, T., Windahl, J., Reutersvärd, P., "Real-Time HWIL simulation of Liquid Food Processor Lines", Modelica conference 2008
- [3] FuelCellLib by Modelon, [www.modelon.com/products/modelica-libraries/fuel-cell-library/](http://www.modelon.com/products/modelica-libraries/fuel-cell-library/)
- [4] Andersson, D., Åberg, E., Eborn, J., Yuan, J., Sundén, B., "Dynamic modeling of a solid oxide fuel cell system in Modelica", Modelica conference 2011
- [5] McBride, B.J., Zehe, M.J., Gordon, S., *NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species*, NASA/TP—2002-211556

# ThermoCycle: A Modelica library for the simulation of thermodynamic systems

Sylvain Quoilin<sup>1</sup>, Adriano Desideri<sup>1</sup>, Jorrit Wronski<sup>2</sup>, Ian Bell<sup>1</sup> and Vincent Lemort<sup>1</sup>

<sup>1</sup> University of Liège, Energy Systems Research Unit

Bâtiment B49, Chemin des Chevreuils 7, 4000 Liège, Belgium

<sup>2</sup> Department of Mechanical Engineering, Technical University of Denmark

Nils Koppels Allé 403, 2800 Kgs. Lyngby, Denmark

## Abstract

This paper presents the results of an on-going project to develop ThermoCycle, an open Modelica library for the simulation of low-capacity thermodynamic cycles and thermal systems. Special attention is paid to robustness and simulation speed since dynamic simulations are often limited by numerical constraints and failures, either during initialization or during integration. Furthermore, the use of complex equations of state (EOS) to compute thermodynamic properties significantly decreases the simulation speed. In this paper, the approach adopted in the library to overcome these challenges is presented and discussed.

*Keywords:* Thermodynamic systems, numerical methods, simulation speed, robustness

## 1 Introduction

Dynamic simulation of thermodynamic systems is required to evaluate and optimize their response time, or to define, implement and test control strategies. [13, 6, 22]. The Modelica language is well adapted to the formulation of thermo-flow problems, mainly because it is an a-causal language that allows interconnecting the models in a "physical" way [5]. In recent years, several libraries have been developed to model thermodynamic and thermal systems in the Modelica language [6, 27]. A number of libraries are now available to model steam and gas cycles (e.g. ThermoSysPro, Power Plants, Thermal Power, ThermoPower, etc.) or refrigeration systems (TIL, AirConditioning, etc.).

However, not all of them are open-access, and few are able to handle non-conventional working fluids such as refrigerants or fluids used in Organic Rankine Cycle (ORC) systems. Thermophysical substance

properties of (moist) air and water are indeed well known and implemented in most simulation tools while those of organic fluids require complex equations of state available only in external libraries such as FluidProp [10], Refprop [16] or CoolProp [1]. Two common solutions when modelling thermodynamic systems requiring external computation of working fluid properties include TIL and ThermoPower:

- TIL is a commercial library for steady-state and transient simulation of thermodynamic systems [26]. The thermodynamic properties are obtained through TILMedia, a library for the calculation of thermophysical substance properties, for example using custom high performance EOS, fast table based bicubic spline interpolation or via an interface for Refprop. It should be noted that TILMedia is not designed according to the Modelica Media standard but provides an interface with the Modelica.Media library. The TIL library includes a variety of models for thermodynamic components (e.g. heat exchangers, expanders, pumps, etc.).
- The ThermoPower library has proven to be well suited for the modelling of power plants, including ORC systems [8]. It can be coupled to the ExternalMedia library [9], an interface between Modelica and, among others, FluidProp. ThermoPower and ExternalMedia are open-source, FluidProp is freely available but not open-source. The models are mainly designed for large-scale power plants with tube heat exchangers. Smaller systems using e.g. plate heat exchangers are less adapted to the ThermoPower library.

The ThermoCycle Modelica library has been developed for the simulation of thermal plants (heat pumps, steam and gas cycles, etc.) with a focus on smaller-capacity systems. It aims at addressing three typical

challenges inherent to the modelling of thermo-flow systems:

1. Computing the thermophysical substance properties of working fluids
2. Computational efficiency
3. Robustness during initialization and integration

Several numerical methods have been developed and implemented in order to enhance the robustness and the simulation speed of the models during initialization and integration. Furthermore, the computation of the working fluid thermophysical properties is achieved by a strong coupling with the open-source thermodynamic properties database Coolprop [1]. The interface between CoolProp and Modelica is based on the Coolprop2Modelica library, a modified version of the ExternalMedia library [9].

## 2 The ThermoCycle Modelica library

The ThermoCycle library aims at providing a robust framework to model small-capacity thermodynamic cycles. The goal is to provide an integrated and fully open-source solution ranging from the thermophysical substance properties, using CoolProp, to the simulation of complex systems with their control strategy. In comparison with alternative libraries dedicated to power plants (ThermoPower, ThermoSysPro, Thermal Power), the ThermoCycle library includes various models dedicated to the modelling of smaller-scale thermal systems, such as volumetric compressors models used for the simulation of heat pump or refrigeration cycles. The key features of the Library are the following:

- Designed for system level simulations
- Full compatibility (connector-wise) with libraries such as MST or ThermoPower, use of stream connectors
- Ability to handle reverse flows and flow reversals
- Various numerical robustness strategies implemented in the components and accessible through Boolean parameters
- High readability of the models (limited levels of hierarchical modelling)

The components provided in the library are designed to be as generic as possible. For example, no detailed

geometry records should be provided by the user to compute the heat exchanger heat transfer and pressure drop. Instead, the user is expected to build a separate detailed model of the component and extract simplified laws to be used in the Modelica component: for a given fluid and operation conditions range, detailed heat transfer correlations can usually be replaced by simplified equations depending on the flow rate only:

$$h = h_{nom} \cdot \left( \frac{\dot{M}}{\dot{M}_{nom}} \right)^n \quad (1)$$

where the  $n$  exponent depends on the flow regime (typically 0.6 for laminar and 0.8 for turbulent). This approach aims at increasing the computational efficiency, and also allows using the same model for different types of components. As an example, the same model can readily be used for shell & tubes or plate heat exchangers only by modifying the  $h_{nom}$  and  $n$  parameters of the simplified heat transfer law.

The same applies for the pressure drops, which can usually be lumped in a single pressure drop on the vapor lines and expressed as a quadratic function of the flow rate. This approach increases the computational efficiency and avoids specific models suitable for one particular component type or geometry only. It is illustrated in Figure 1 for the case of a heat pump: the pressure drops in each line are lumped into two pressure drop components that were calibrated in a separate and more detailed steady-state model. It should be noted that this assumption is valid only if the pressure drops remain limited in both heat exchangers: their influence on the temperature profile should be small compared to the temperature differences and the pinch points of the process.

### 2.1 Modelling of fluid flows

The finite volume approach is selected as general method to simulate fluid flows. Homogeneous one- and two-phase flow are taken into account, while non-homogeneous, multi-phase flows are not considered, yet. The basic fluid flow component is a cell in which the energy and mass conservation equations are applied (Figure 2). Two types of variables can be distinguished: cell variables and node variables. Node variables are distinguished by the "su" (supply) and "ex" (exhaust) subscripts, and correspond to the inlet and outlet nodes of each cell (Fig. 2). The relation between the cell and node values depends on the discretization scheme. Two schemes are implemented in the cell component, the central difference scheme ( $h = (h_{su} + h_{ex})/2$ ) and the upwind scheme ( $h = h_{su}$ ).

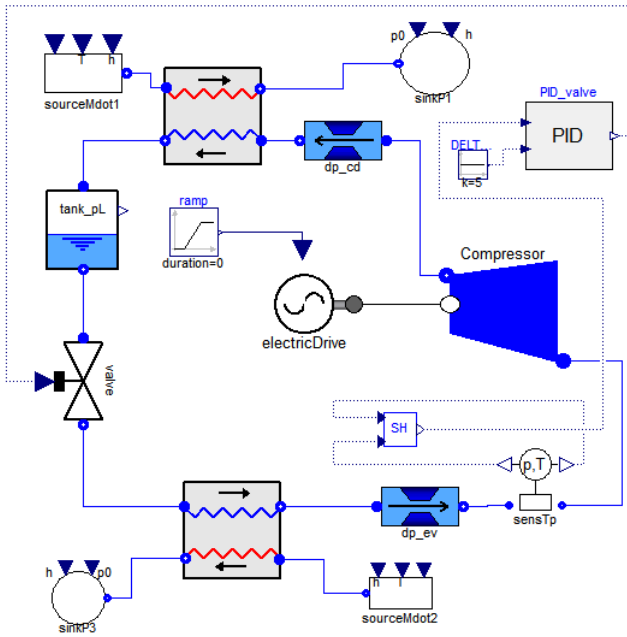


Figure 1: Model of an R407c heat pump

Since the model accounts for flow reversal, a conditional statement is added depending on the flow rates at the inlet and outlet nodes. For the central difference scheme,  $h_{su}$  is expressed by Equation 2 (an equivalent equation applies to  $h_{ex}$ ):

$$h_{su} = \begin{cases} h_{ex}^* & \text{if } \dot{M}_{su} \geq 0 \\ 2 \cdot h - h_{ex} & \text{if } \dot{M}_{su} < 0 \text{ and } \dot{M}_{ex} < 0 \\ h & \text{if } \dot{M}_{su} < 0 \text{ and } \dot{M}_{ex} \geq 0 \end{cases} \quad (2)$$

where the flow rates are defined as positive when the fluid flows in the nominal direction (from "su" to "ex"), and where  $h_{ex}^*$  indicates the exhaust node enthalpy of the previous cell.

For the upwind scheme:

$$h_{su} = \begin{cases} h & \text{if } \dot{M}_{su} < 0 \\ h_{ex}^* & \text{if } \dot{M}_{su} \geq 0 \end{cases} \quad (3)$$

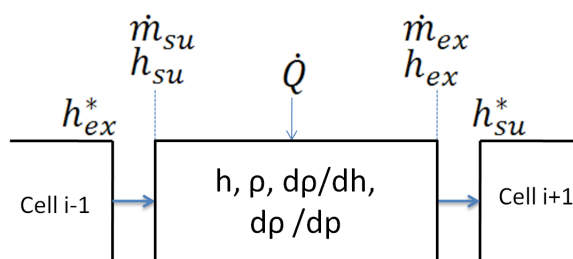


Figure 2: Discretized flow model with cells and node variables

The energy balance for one cell is expressed by [21]:

$$V\rho \frac{dh}{dt} = \dot{M}_{su}(h_{su} - h) - \dot{M}_{ex}(h_{ex} - h) + \dot{Q} + V \frac{dp}{dt} \quad (4)$$

and the mass balance is written as a function of the two differentiated state variable  $p$  and  $h$

$$\dot{M}_{ex} - \dot{M}_{su} = V \cdot \frac{dp}{dt} = V \cdot \left( \frac{\partial \rho}{\partial h} \cdot \frac{dh}{dt} + \frac{\partial \rho}{\partial p} \cdot \frac{dp}{dt} \right) \quad (5)$$

where  $\frac{\partial \rho}{\partial h}$  and  $\frac{\partial \rho}{\partial p}$  are considered as thermodynamic properties of the working fluid and are computed in CoolProp. Pressure is considered constant in the cell component.

The cell model has been developed for compressible, incompressible, and constant heat capacity fluid. The overall flow model can be obtained by connecting several cells in series. The final discretization scheme corresponds to a staggered grid, i.e. the thermodynamics states and the state variables (the enthalpies) are calculated inside the cells and the node values ("su" and "ex") are deduced using equations 2 and 3.

## 2.2 Library structure

The library is organized into different sub packages, including:

1. *Components*, is the central part of the library. It is divided in three sub packages: FluidFlow, Heat-Flow and Units. It contains all the models available in the library from the simple cell model for fluid flow to complete models of heat exchangers, expanders and control units. A more detailed description of this package is presented in subsection 2.3.
2. *Examples*, contains simulation models where the components of the library are tested. It includes several ORC plant models and it also provides a step by step package where the procedure to build an ORC power unit and an heat pump system is described in detail.
3. *Functions*, includes the empirical correlations used to characterize some of the library models as well as general purpose mathematical functions.
4. *Interfaces*, contains the connectors used for the library components.
5. *Media*, predefines a list of the fluids available in the library.

Figure 3 shows an overview of the library structure.

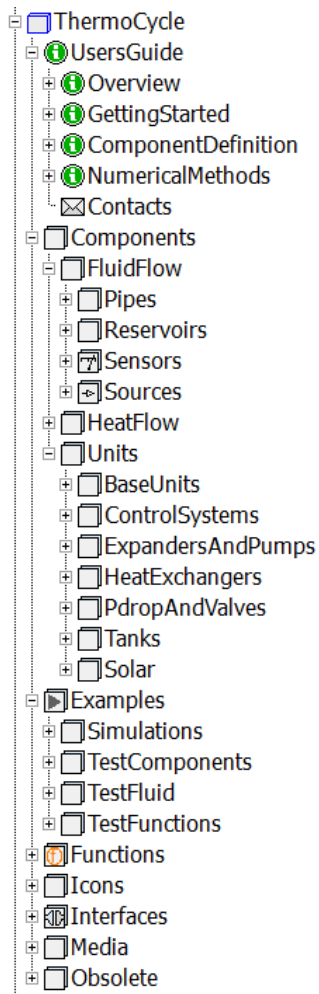


Figure 3: An overview of the library structure from the Dymola graphical user interface

## 2.3 Physical components

The Components package of the library provides a set of models from the basic cell model to higher-level components commonly used in thermal systems. The most common ones are described hereunder.

### 2.3.1 Expansion and compression machines

Since the time constants characterizing expansion and compression processes are small compared to that of the heat exchangers [14], the basic pump expander and compressor models are described by empirically derived algebraic correlations. In particular, pumps are modelled by two empirical equations expressing the isentropic and volumetric efficiencies as a function of the operating conditions, e.g. rotational speed and pressure ratio.

Low capacity systems generally use volumetric expansion machines instead of turbomachinery. Therefore,

special attention has been paid to their modelling in ThermoCycle. Advanced empirical expressions of the efficiency and filling factor are used and implemented, such as the one proposed by Declaye et al. in the case of an open-drive scroll expander [11].

In the case of volumetric compressors, the efficiencies (isentropic, volumetric) can either be user-defined or based on the standard EN12900, which is a well known standard to express the compressor performance as a polynomial function of the evaporation and condensation temperatures.

In addition to these empirical models, more detailed physical models are also available, such as the one proposed by Lemort et al. [17] for scroll machines which takes into account thermal losses, friction losses, internal leakage, internal pressure drop or under and over-expansion (Figure 4). Finally, detailed models of the expansion machine have been included, in which the thermodynamic state of the fluid in the expansion chambers is computed throughout one whole revolution [28]. This type of model is well adapted to the optimization and to the simulation of the component alone, but can hardly be integrated into a more general model because of its significant computational time.

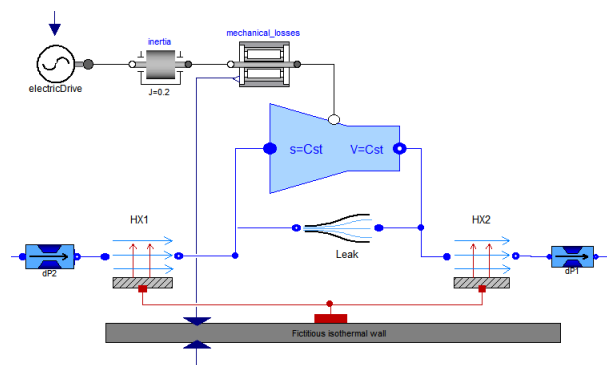


Figure 4: Semi-empirical compressor model

### 2.3.2 Heat exchangers

The base component for all heat exchangers is the cell model, see Figure 2. Two types of cells are distinguished: one for compressible (and two-phase) flows, and one for incompressible flows. Heat exchanger models are built by interconnecting these cells in the appropriate manner together with a wall model [27]. Parallel, counter flow and cross flow models for compressible and incompressible fluids are available. More complex structure can also easily be developed by modifying the interconnections between the different fluid flow models.

In addition to the above models based on the finite volumes method, a moving boundaries heat exchanger model for evaporators has also been included. This type of model is known to be more computationally efficient than discretized models and is less likely to be subject to chattering.

### 2.3.3 Pressure drop and valves

Pressure drops are modelled by lumped parameters used to compute the pressure difference as a function of the flow rate. This function includes a constant term (to model static pressure differences), a linear term (e.g. to model laminar pressure losses) and a quadratic term (to model turbulent pressure losses):

$$\Delta p = \rho \cdot g \cdot h + K \cdot \dot{V} + \frac{1}{A^2} \cdot \frac{\dot{M}^2}{2 \cdot \rho} \quad (6)$$

where  $\rho$  is the fluid density,  $g$  is the gravitational constant,  $\dot{V}$  is the volume flow rate,  $\dot{M}$  is the mass flow rate. The three parameters are  $h$  (the static head),  $K$  (the linear pressure drop coefficient) and  $A$  (the equivalent turbulent orifice area).

Infinite derivative at zero flow in the case of the quadratic term is avoided by using the Modelica *regSquare* function. The fluid is assumed to be incompressible and ambient losses are neglected.

Valves are modelled as a pressure drop whose orifice cross-sectional area can be adjusted by an external signal.

### 2.3.4 Tanks and liquid receivers

Different tank (fully mixed) and liquid receiver models are available. They are modelled using the energy and mass conservation principles, and assuming thermodynamic equilibrium at all times inside the control volume.

### 2.3.5 Solar field

In recent years, several studies have underlined the potential of small-capacity thermal solar organic power systems [19, 18, 20]. Different models of parabolic trough solar collector have therefore been added to the library. The large ratio between diameter and length allows a one dimensional discretization of the absorber tube.

The model is composed by two subcomponents, as shown in figure 5: the Flow1Dim component which models the heat transfer fluid flow in the heat collector element (HCE) and the SolAbs component

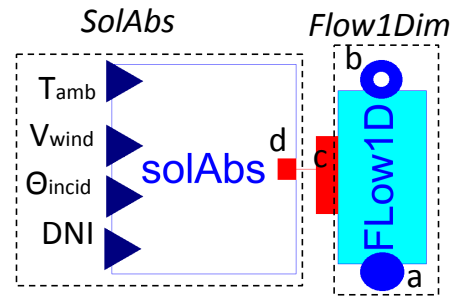


Figure 5: Object diagram of the solar field model. a and b indicate respectively the fluid inlet and outlet ports of the flow model (Flow1Dim). c and d indicate the thermal ports.

which solves the dynamic one dimensional energy balance around the HCE. The solAbs component implements the relations between the environmental parameters, direct normal radiation (DNI), incidence angle ( $\Theta_{\text{incid}}$ ), ambient temperature ( $T_{\text{amb}}$ ), wind velocity ( $v_{\text{wind}}$ ) and the axial temperature distribution along the absorber, either based on the Forristal model [12] or based on an empirical correlation derived from the Schott test analysis [3]. The user can choose between different collector geometries based on commercial products. The solar field model is available for compressible and incompressible fluids.

## 3 The CoolProp2Modelica library

The ThermoCycle library aims at modelling different kinds of thermodynamic cycles, including those with unconventional working fluids. The efficient computation of their thermophysical properties is therefore a key aspect of the proposed models. As previously mentioned, the currently available Modelica solutions for computing organic fluid properties are limited to non-open-source databases. In order to propose a fully open-source tool, ThermoCycle has been coupled to CoolProp, an open-source library developed at the University of Liège [1]. The main features of the CoolProp library are the following:

- Fully open-source
- High accuracy Helmholtz energy-based equations of state.
- More than 110 different working fluids
- Properties over 40 incompressible fluids (e.g. thermal oils) and brines (only accessible with p,T or p,h as input variables).

- Low computational time

The interface between Modelica and CoolProp is implemented in the CoolProp2Modelica library, a modified version of the ExternalMedia library [9].

In thermo-flow systems, the computation of the thermophysical properties amounts for a significant share of the total computational time. In order to enhance fluid property calculation speed, two interpolation methods, the tabular Taylor series expansion (TTSE) and the bicubic interpolation method, have been developed and integrated into the CoolProp main source code [1]. The interpolation tables (the working fluid properties computed on a 200x200 grid over the whole range of states) are built at the beginning of the simulation (at the first property call) and stored in memory for further use. This process requires between 2 s and 10 s. All subsequent property calls are performed using the selected interpolation method. It should be noted that, contrary to the bicubic method, the TTSE method generates discontinuities in the computation of the fluid properties. This discontinuity is however very small and does not constitute an issue during simulation.

In order to assess the effectiveness of the developed methods, a benchmark test was carried out comparing the different available fluid properties database with respect to the elapsed time per property call from Modelica. The properties were called a large amount of times and in different areas of the thermodynamic diagram (liquid, two-phase and vapor). The results are presented in figure 8 and show that Coolprop is significantly faster than Fluidprop and slightly slower than the commercial library TILMedia/RefProp. However, when the TTSE method is activated, the computational time is one order of magnitude lower than TILMedia, demonstrating the efficiency of the proposed interpolation method.

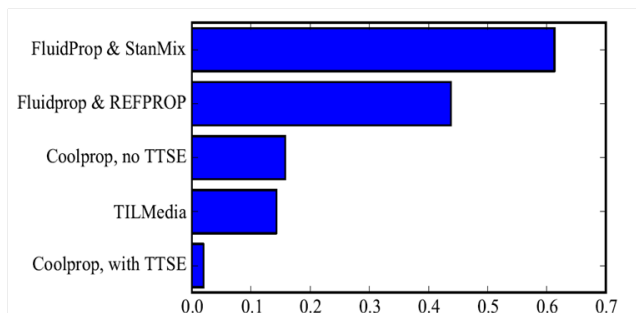


Figure 6: Elapsed time per property call from Modelica in the case of propane (in ms)

In addition to the interpolation methods, smoothing methods to deal with the discontinuity of the density derivatives in the two-phase zone, described in section 4, have also been implemented. The open-source characteristics of CoolProp is a non-negligible advantage since it allows the process to be fully transparent for the end-user, who only needs to activate a flag in the Modelica property call, as shown in figure 7. In this example, the fluid name is modified to instruct CoolProp not to compute the transport properties, the debug level is set to unity, the density is smoothed out in the vapor quality interval from 0 to 0.15, and the TTSE interpolation method is activated.

```

package R245faCoolSmooth
extends CoolProp2Modelica.
Interfaces.ExternalTwoPhaseMedium(
  mediumName = "R245fa",
  libraryName = "CoolProp",
  substanceNames = {"R245fa|calc_transport=0|debug=1|
rho_smoothing_xend=0.15|enable_TTSE=1"});
end R245faCoolSmooth;

```

Figure 7: Modelica code to define a fluid. A flag for a numerical and an interpolation method is activated

## 4 Numerical aspects

Dynamic modelling of thermodynamic cycles can be a challenging task, among others because of the numerous numerical issues arising both during initialization and during integration. In order to enhance the performance and the robustness of the ThermoCycle library, different numerical methods have been implemented and tested. They are presented and discussed in this section.

### 4.1 Initialization

The convergence of the Newton Solver during initialization is a key challenge when modelling complex system. Several strategies have been developed, such as the *homotopy* method [25]. In addition to this method, a slightly different approach is proposed in ThermoCycle: the system is initialized on a simplified system of equations, and the more complex non-linear equations, such as the computation of the heat transfer coefficients as a function of mass flow, are activated one by one during integration using an *initialization* component developed for that purpose.



## 4.2 Chattering and flow reversals

The phenomenon of chattering may occur when discontinuities in the model variables are present [15]. This phenomenon can lead to extremely slow simulation, or to simulation failures because the computed variables exceed acceptable boundaries. In discretized two-phase flow models, the main discontinuity often occurs in the density derivative on the liquid saturation curve. Simulation failures or stiff systems can occur if the cell-generated (and purely numerical) flow rate causes a flow reversal in one of the nodes due to this discontinuity. The computation of  $h_{su}$  and  $h_{ex}$  switches from one value to the other in equations 2 and 3.

In addition to chattering, flow reversals can also result in a singular and non-solvable set of equations, as shown in [24]. A solvability criterion can be expressed as

$$h_{su} > h + \frac{\rho}{\frac{\partial \rho}{\partial h}}, \quad (7)$$

where  $\partial \rho / \partial h$  is a negative term.

This inequality states that in cases of flow reversal, an unsolvable system of equations appears, if the enthalpy of the entering fluid is below a certain limit. A formal demonstration of this effect can be found in [24].

Therefore, to ensure the robustness of the simulation and to avoid chattering or unsolvable systems, two strategies can be employed:

1. Avoid flow reversals caused by the density derivative discontinuity, see Eq. 5.
2. If flow reversal occurs (it is physically possible), make sure that the backward flow enthalpy is higher than the limit described in Eq. 7.

The first strategy can be expressed by an inequality stating that purely numerical, cell-generated flow rates must be lower than the flow rate circulating through the cycle, which can be written (for a single cell):

$$\dot{M}_{ext} \gg \frac{V}{N} \cdot \frac{d\rho}{dt} = \frac{V}{N} \cdot \left( \frac{\partial \rho}{\partial h} \cdot \frac{dh}{dt} + \frac{\partial \rho}{\partial p} \cdot \frac{dp}{dt} \right) \quad (8)$$

According to Eq. 8, flow reversals and thus chattering or simulation failures are likely to occur if:

- The number of cells ( $N$ ) is low
- The working fluid flow rate ( $\dot{M}_{ext}$ ) is low
- The internal volume ( $V$ ) is high
- The working conditions are highly transient (i.e.  $dp/dt$  and  $dh/dt$  are high)

Different methods are implemented in ThermoCycle to avoid the simulation issues described above. Some are implemented at the Modelica level while others require a modification of the thermodynamic properties of the working fluid and are therefore implemented into CoolProp. It should also be noted that some of these methods have already been proposed in the literature, while some others are new. They are briefly described hereunder. A more comprehensive description is available in [23].

- *Filtering method*: In this strategy, a first order filter is applied to the fast variations of the density with respect to time. This filter therefore acts as "mass damper" and avoids transmitting abrupt variations of the flow rate due the density derivative discontinuity.
- *Truncation method*: This strategy acts on the terms  $\partial \rho / \partial p$  and  $\partial \rho / \partial h$  of Eq. 8. The peak in the density derivative occurring after the transition from liquid to two-phase is truncated, reducing the numerical flow rate generated by the density derivative discontinuity.
- *Smoothing of the density derivative*: The idea behind this method is to smooth out the density derivative discontinuity using a spline function. Modifications of the thermophysical properties are implemented at the level of the equation of state, i.e. inside the CoolProp database. The main drawback of this method is that the density function is still calculated with the original equation of state: the smoothed density derivative is not consistent with the density function provided by the EOS. This might cause a mass defect during the simulation.
- *Smoothing of the density function*: In order to avoid the mismatch between the density function and its derivative, one possible solution is to smooth the density for a range of vapour qualities (i.e. making it C1-continuous) and recalculating its partial derivatives in the smoothed area. In this situation, the density derivatives are continuous but not smooth, which should still be manageable for the solver.
- *Mean densities method*: The mean densities method was originally proposed by Casella [4] and successfully tested by Bonilla et al. [2]. It is also the method implemented in the ThermoPower Modelica library [7]. A mean density and its partial derivatives are computed in each cell as a function of the node densities, which

eliminates the discontinuity in the partial derivatives.

- *The enthalpy limiter method*: Contrary to the previous methods, the enthalpy limiter method does not aim at avoiding flow reversals. Instead, it ensures that the system of equations remains solvable even in case of flow reversal. As indicated in Eq. 7, the enthalpy of the fluid entering a cell should have a minimum value, ensuring that the system of equations can be solved. The enthalpy limiter method is the practical implementation of this constraint in the cell model. It was originally proposed by Schulze et al. [24] and implemented in the TIL Modelica library.
- *Smooth Reversal Enthalpy*: In case of flow reversal a discontinuity appears in the computed node enthalpy. In this method, this is solved using a smooth transition function for the computation of the enthalpy as a function of the flow rate close to zero. The main drawback is the generation of a highly nonlinear algebraic system that has to be solved by the simulator.

A comparison of these different methods, based on the simulation of a flow model with a high number of cells, in terms of simulation speed, simulation accuracy and mass and energy imbalance shows that adopting the proposed methods can dramatically improve the simulation performance and even allow to simulate flow reversals where traditional models generate simulation failures. The main concern is the error they introduce in the simulation results and the possible mass and energy unbalances they generate. In [23], the methods have been tested on a test system submitted to highly transient conditions. Results have shown that although numerical artifacts are generated, the error remains of the same order of magnitude as the error linked to the standard finite volume model with a  $10^{-4}$  tolerance. This is an important statement showing that the proposed models can ensure failure-free simulation even in highly transient conditions. It should also be noted that those transient conditions are likely to generate chattering or stiff systems are usually concentrated in specific times of the simulation (e.g. start-up and shut-down), in which robustness is more important than accuracy.

## 5 ThermoCycle Viewer

When debugging or interpreting simulation results, it is generally useful to post-process the raw time-data

before displaying it. To that end, an analysis tool has been developed, which scans the simulation results and automatically detects all thermodynamic states to display them on a thermodynamic diagram (e.g. T-s or p-h). The heat exchanger temperature profiles are also detected, displayed, and animated as a function of time.

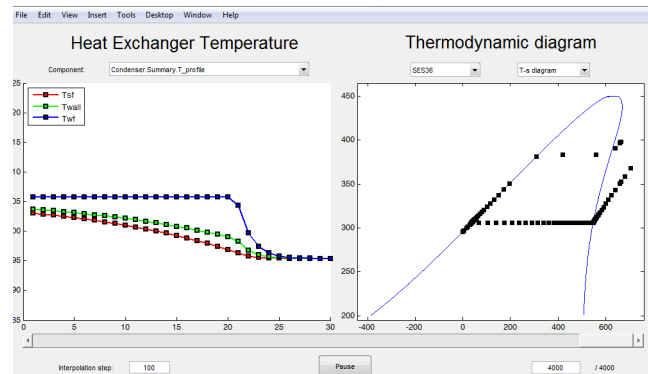


Figure 8: Graphical interface to display thermodynamic states and temperature profiles

## 6 Conclusion

The development of the ThermoCycle library is an on-going process aiming at providing a completely open-source tool for the dynamic modelling of thermodynamic cycles. The library comprises a number of components which can be used to simulate a wide range of physical systems. Special attention has been paid to the implementation of new and existing numerical methods to enhance the robustness and the simulation speed of the models during initialization and integration. The ThermoCycle and CoolProp2Modelica libraries are being released together with ThermoCycle Viewer through the library website (URL: [www.thermocycle.net](http://www.thermocycle.net)). Current work now focuses on the integration of new components (moving boundaries models, thermocline and stratified storage, evaporator with frosting, etc.) and on the experimental validation of the proposed models. At the present time, the library and the thermophysical properties database are both open-source, but are running exclusively under Dymola. Porting the library to the OpenModelica platform is another ongoing work, which should lead to a fully integrated open-source solution for the modelling of thermal systems.

## 7 Acknowledgement

Some results presented in this paper have been obtained within the frame of the IWT SBO-110006 project The Next Generation Organic Rankine Cycles ([www.orcnex.be](http://www.orcnex.be)), funded by the Institute for the Promotion and Innovation by Science and Technology in Flanders. This financial support is gratefully acknowledged.

## References

- [1] Ian H. Bell, Jorrit Wronski, Sylvain Quoilin, and Vincent Lemort. Pure- and Pseudo-Pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp. *Industrial & Engineering Chemistry Research*, 2014.
- [2] J. Bonilla, L. J. Yebra, and S. Dormido. Mean densities in dynamic mathematical two-phase flow models. *Computer Modeling in Engineering and Sciences (CMES)*, 67(1):13, 2010.
- [3] F. Burkholder and C. Kutscher. Heat loss testing of schott's 2008 ptr70 parabolic trough receiver. Technical report, NREL, 2009.
- [4] F. Casella. Object-oriented modelling of two-phase fluid flows by the finite volume method. *Proceedings 5th Mathmod Vienna, Austria, Sep*, 2006.
- [5] F. Casella, J.G. van Putten, and P. Colonna. Dynamic simulation of a biomass-fired steam power plant: A comparison between causal and a-causal modular modeling IMECE2007-41091. In *Proceedings of the International Mechanical Engineering Congress*, volume 6, pages 205–216, 2007.
- [6] Francesco Casella and Alberto Leva. Modelica open library for power plant simulation: design and experimental validation. In *Proceeding of the 2003 Modelica conference, Linkoping, Sweden*, 2003.
- [7] Francesco Casella and Alberto Leva. Modelica open library for power plant simulation: design and experimental validation. In *Proceeding of the 2003 Modelica conference, Linkoping, Sweden*, 2003.
- [8] Francesco Casella, Tiemo Mathijssen, Piero Colonna, and Jos van Buijtenen. Dynamic modeling of organic rankine cycle power systems. *Journal of Engineering for Gas Turbines and Power*, 135(4):042310, March 2013.
- [9] Francesco Casella and Christoph Richter. External media: A library for easy re-use of external fluid property code in madelica. In *In proceeding of the 6th International Modelica Conference*, 2008.
- [10] P. Colonna and T. P. van der Stelt. *FluidProp: a program for the estimation of thermo physical properties of fluids*. Energy Technology Section, Delft University of Technology, 2004.
- [11] Sebastien Declaye, Sylvain Quoilin, Ludovic Guillaume, and Vincent Lemort. Experimental study on an open-drive scroll expander integrated into an {ORC} (organic rankine cycle) system with {R245fa} as working fluid. *Energy*, 55(0):173 – 183, 2013.
- [12] R. Forristall. Heat transfer analysis and modeling of a parabolic trough solar receiver implemented in engineering equation solver. Technical Report Task No. CP032000, National Renewable Energy Laboratory (U.S. Department of Energy), 2003.
- [13] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons, August 2010.
- [14] L. Guangbin, Z. Yuanyang, L. Yunxia, and L. Liansheng. Simulation of the dynamic processes in a scroll expander-generator used for small-scale organic rankine cycle system. In *Proceedings of the institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 2011.
- [15] Jakob Munch Jensen. *Dynamic Modeling of Thermo-fluid Systems: With Focus on Evaporators for Refrigeration : Ph.D.-thesis*. Energy Engineering, Department of Mechanical Engineering, Technical University of Denmark, 2003.
- [16] E.W. Lemmon, M.L. Huber, and M.O. McLinden. *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP*. National Institute of Standards and Technology, Boulder, Colorado, 2010.

- [17] Vincent Lemort, Sylvain Quoilin, Cristian Cuevas, and Jean Lebrun. Testing and modeling a scroll expander integrated into an organic rankine cycle. *Applied Thermal Engineering*, 29(14-15):3094–3102, 2009.
- [18] E. Prabhu. Solar trough organic Rankine electricity system (STORES) stage 1: Power plant optimization and economics. Technical Report NREL/SR-550-39433, National Renewable Energy Laboratory, 2006.
- [19] H. Price and V. Hassani. Modular trough power plant cycle and system analysis. Technical Report NREL/TP-550-31240, National Renewable Energy Laboratory, 2002.
- [20] S. Quoilin, M. Orosz, H. Hemond, and V. Lemort. Performance and design optimization of a low-cost solar organic rankine cycle for remote power generation. *Journal of Solar Energy Engineering*, 85:955–966, 2011.
- [21] Sylvain Quoilin. *Sustainable Energy Conversion Through the Use of Organic Rankine Cycles for Waste Heat Recovery and Solar Applications*. PhD thesis, University of Liege, Belgium, 2011.
- [22] Sylvain Quoilin, Richard Aumann, Andreas Grill, Andreas Schuster, Vincent Lemort, and Hartmut Spliethoff. Dynamic modeling and optimal control strategy of waste heat recovery organic rankine cycles. *Applied Energy*, 88(6):2183–2190, 2011.
- [23] Sylvain Quoilin, Ian Bell, Adriano Desideri, and Vincent Lemort. Methods to increase the robustness of finite-volume flow models in thermodynamic systems. *Energies*, 2014.
- [24] Christian Schulze, Manuel Graber, and Wilhelm Tegethoff. A limiter for preventing singularity in simplified finite volume methods. In *Mathematical Modelling*, 2012.
- [25] Michael Sielemann, Francesco Casella, Martin Otter, Christoph Clauß, Jonas Eborn, Sven Erik Mattsson, and Hans Olsson. Robust initialization of differential-algebraic equations using homotopy. In *Proceedings of the 8th Modelica Conference*, pages 21–22, Dresden, 2011.
- [26] TLK Thermo GmbH. TIL Suite - Simulates Thermal Systems, April 4th 2013.
- [27] Hubertus Tummescheit, Jonas Eborn, and Falko Jens Wagner. Development of a modelica base library for modeling of thermo-hydraulic systems. In *Proceedings of the Modelica Workshop 2000*, 2000.
- [28] Jorrit Wronski, Jean-Francois Oudkerk, and Fredrik Haglind. Modelling of a small scale reciprocating ORC expander for cogeneration applications. In *ASME-ORC 2013 – 2nd International Seminar on ORC Power Systems*, 2013.

# An Operational Semantics for Hybrid Systems Involving Behavioral Abstraction

Simon Bliudze  
École Polytechnique Fédérale de Lausanne  
INJ Building, Station 14  
1015 Lausanne, Switzerland  
simon.bliudze@epfl.ch

Sébastien Furic  
LMS Imagine S.A.  
7 place des Minimes  
42300 Roanne, France  
sebastien.furic@lmsintl.com

## Abstract

We discuss the challenges of building a simulation framework for hybrid systems, in particular the well-known Zeno effect and correct composition of models idealised by abstracting irrelevant behavioural details (e.g. the bounce dynamics of a bouncing ball or the process of fuse melting in an electrical circuit). We argue that the cornerstone of addressing these challenges is the definition of a semantic framework with an appropriate underlying model of time.

Using two simple examples, we illustrate the properties of such a model and explain why existing models are not sufficient. Finally, we propose a new Zeno-free semantic model that allows mixing discrete and continuous behaviour in a rigorous way and provides for the compositional behavioural abstraction.

Although it is based on non-standard analysis, we explain how our semantic model can be used to develop hybrid system simulators.

*Keywords:* Hybrid Modeling Languages; Non-Standard Analysis; Models of Signals; Behavioral Abstraction; Operational Semantics

## 1 Introduction

A large number of modelling, verification and simulation frameworks for hybrid systems have been designed in the past years. Although, a complete overview is beyond the scope of our paper, we observe that they broadly fall in two categories: those that put special emphasis on a rigorous model definition, such as, for instance, the Ptolemy project [6] (based on [14]), the Zélus synchronous language [3] (based on the semantics in [1]) and SpaceX [7]; and those that have chosen a more pragmatic, informal approach, such as the Modelica language [8] and the as-

sociated tools, and the Scicos block-diagram modeller and simulator [4].

All the associated tools share the same basic model of execution alternating between continuous phases and sequences of ‘run-to-completion’ discrete actions [3] as formalised by the notion of hybrid automata [11]. None of these approaches attempts to include the operational semantics of differential equations in their core semantic model: execution of the continuous phases is delegated to numerical solvers, which are used to advance physical time and compute the values of physical signals.

Except for Zélus, none of the above semantic models are Zeno-free, which means that, as explained in the next sections, they do not reflect the fact that time diverges and rely on analysing the solver output to detect and advance past the Zeno points [13] (cf. Section 2). This poses a fundamental problem, since the solver behaviour at this point is usually unspecified.

Furthermore, none of the above proposals allows *compositional behavioural abstraction*: idealised models do not account for the physical nature of phenomena, in particular the fact that original, *high-fidelity* signals are *continuous*. However, this property is assumed by most users and validated by most real-life systems. Hence, it must be a fundamental property of signals and should be reflected in their idealisations.

In our view, to achieve maximum robustness of a simulation framework, it is crucial to define the semantic model before designing either the language or the simulator. Thus, the design of a hybrid simulation framework should involve the following steps.

First of all, one must define a semantic model that properly accounts for the expected elementary properties of systems to be simulated. This includes dynamic behaviour properties, but also “higher level” ones, such as modularity.

The second step consists in designing a simulator

capable of computing acceptable approximations of the dynamics of models conforming to the semantic model above. Successful completion of this step validates the semantic model by showing that all conforming models can be simulated.

The third step involves the design of a language powerful enough to express a useful subset of models that provably conform to the semantic model. In order to avoid errors at simulation time or, worse, spurious simulation results, the semantic validity of a model must be provable statically, e.g. by type-checking.

Finally, once both the semantic model and the language have been defined, one has to design a compiler for this language, which would perform the necessary static checks and reject the invalid models.

In this paper, we focus on the first step above, i.e. defining the appropriate semantic model. Indeed, in our opinion, current semantic models still lack some of the properties required by real-life systems.

In particular, modularity is an extremely important property for both correctness and practical usability of hybrid simulation frameworks. One of us has already discussed some modularity issues in Modelica [9]. In the present paper, we focus on another—probably the most important—aspect contributing to modularity, which is *compositional behavioural abstraction*.

Realistic hybrid models almost always require part of the physical behaviour to be abstracted by means of “ideal equations”—such as conditional and reset equations—that typically yield discontinuities in physical signals. Below, we will use the term *behavioural abstraction* more generally to designate any mechanism that enables concrete physical behaviour to be “hidden” by considering *idealised* models. Intuitively, from the point of view of an external observer, behavioural abstraction makes the model “jump” over instants corresponding to activations of abstraction mechanisms. Thus, idealisation consists in explicitly providing a constraint to be met *after* these instants.

It is well known from physical system modelling experts that, although extremely useful in practice, behavioural abstraction often leads to inconsistencies and even to singularities in simulation code, especially when abstractions are *composed*. Informally, we say that behavioural abstraction is *compositional* if substituting an ideal sub-model—instead of a high-fidelity one—within a larger model does not introduce new behaviours that were not present in the original model (a counter-example is given in Section 3.1).

The cornerstone of a semantic framework that would address the above issues is the underlying

model of time.

Traditional definition of the simulation time-line either relies on a fixed basic step, provided as a simulation parameter, or a variable step adjusted based on events detected during the simulation. The former has the advantage of being Zeno-free by construction, since at each simulation step the time advances by a constant value, but produces wrong simulation results whenever the signal activity is higher than the fixed sampling frequency. Variable step simulation, on the other hand, improves the simulation precision by sacrificing Zeno-freeness. However, in both cases, the semantics of a model is only defined at simulation time depending on the choice of time steps. Hence, model validity cannot be statically proven.

In Section 3.1, we argue that, to ensure compositionality of behavioural abstraction and reflect the intrinsic continuity of physical phenomena, the time model must be *densely ordered*.<sup>1</sup> Indeed, as a result of behavioural abstraction, several causally dependent events happen at the same instant. Traditional time models above do not provide any possibility to preserve the causality information in the semantic model, leading to spurious behaviours.

The *super-dense* time approach [14] addresses this problem by defining time as a subset of the cartesian product  $\mathbb{R} \times \mathbb{N}_0$ . However, solutions of differential equations are obtained by means of standard operational semantics. Thus, although causality between instantaneous events can be preserved, the problems related to the Zeno effect persist.

Recent use of Non-Standard Analysis [12] in the design of operational semantics for hybrid systems [1, 2, 16] has led to the definition of a *linear* time that is 1) *discrete*, i.e. instants can be considered in isolation; 2) *well-ordered*, i.e. for each time instant, there is a uniquely defined *next instant* respecting the usual temporal order; and 3) it can be treated as a *continuum*. These properties make it suitable to express both discrete and continuous dynamics in a *unified* fashion and avoid the Zeno effects. However, as a consequence, such time model cannot be densely ordered. Indeed, it features *consecutive instants*, which, by definition, do not have any instants between them.

In this paper, we propose a new, Zeno-free semantic model also based on a non-standard model of time, which allows mixing discrete and continuous behaviour in a rigorous way. In addition, this new model also allows compositional behavioural abstrac-

<sup>1</sup>A partially ordered set is said to be densely ordered if for all elements  $x$  and  $y$  for which  $x < y$  there exists a  $z$  such that  $x < z < y$ .

tion: density of time is reestablished so that physical signals remain continuous even when idealised, which avoids the emergence of “impossible behaviours”.

The key idea behind our proposal is to discretise the *value* of signals instead of time. Discretisation is necessary in order to associate *dates* with events [1, 2, 16]. However, discretising signal values allows us to choose these dates among all non-standard reals, rather than fixing a regular time-line in advance. Thus, we consider discrete signal *activity* in the frame of a densely-ordered time-line compatible with the requirements of compositional behavioural abstraction. This approach has something in common with [10], where well-ordered time scales are constructed as subsets of a common densely-ordered time reference.

The paper is structured as follows. In Section 2, we explain the need for an operational semantics that can cope with the Zeno paradox, and how non-standard analysis can be used to define such a semantics. In Section 3, we show why previous proposals based on non-standard analysis do not ensure compositional behavioural abstraction, by taking an example involving the *composition* of several abstractions. Finally, in Section 4, we present our proposal, also based on non-standard analysis, but where discretisation is no longer applied to the time-line but to signal values, leading to a more intuitive definition of time that, moreover, allows compositional behavioural abstraction.

## 2 Why non-standard semantics?

### 2.1 Behavioural abstraction example

The famous bouncing ball model, whereof a Modelica implementation is given in Listing 1, is probably one of the simplest models involving behavioural abstraction. In this model, the physics of the bounce have been abstracted away: we consider that, at bounce time, the velocity, which was negative just before the ball hits the ground, *instantaneously* becomes positive, with a magnitude decreased by 20%. Figure 1 shows the simulation results for  $x$  in the  $[0, 1.1]$  time interval.

Despite its apparent simplicity, this model poses severe challenges to language theorists as we shall see.

### 2.2 The Zeno effect

We suppose here that physical time is based on standard reals (i.e.  $\mathbb{R}$ ). How does our model behave with such a time model?

Let’s introduce the following notations:

Listing 1: A bouncing ball model, in Modelica.

```

model BouncingBall
  Real v, x;
  constant Real g = 10;
  initial equation
    v = 1.0;
    x = 0.0;
  equation
    der(v) = -g;
    der(x) = v;
    when x < 0 then
      reinit(v, -0.8 * pre(v));
      reinit(x, 0.0);
    end when;
end BouncingBall;

```

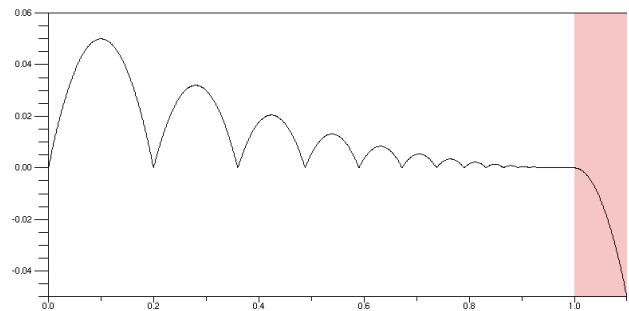


Figure 1: Simulation results for the bouncing ball model with the DASKR solver (altitude of the ball with respect to time)

- $t_i$ , for  $i \in \mathbb{N}_0$ , denotes the initial instant of the  $i^{\text{th}}$  flight of the ball;
- $v_i$ , for  $i \in \mathbb{N}_0$ , denotes the (positive) velocity of the ball at  $t_i$ .

According to the model, the trajectory of the ball in-flight has to verify:

$$\dot{v} = -g \quad (1a)$$

$$\dot{x} = v \quad (1b)$$

Equations (1a) and (1b) can be solved analytically for  $v$  and  $x$ , for  $t \in [t_i, t_{i+1})$ , giving:

$$v(t) = -g \cdot (t - t_i) + v_i \quad (2a)$$

$$x(t) = -\frac{1}{2}g \cdot (t - t_i)^2 + v_i \cdot (t - t_i) \quad (2b)$$

Since at  $t_{i+1}$  the ball hits the ground (that is,  $x = 0$ ), we must have:

$$0 = -\frac{1}{2}g \cdot (t_{i+1} - t_i)^2 + v_i \cdot (t_{i+1} - t_i) \quad (3)$$

From (3) we deduce the duration of the  $i^{\text{th}}$  flight (notice that it only depends on the velocity of the ball at  $t_i$ ):

$$t_{i+1} - t_i = \frac{2}{g}v_i \quad (4)$$

Also, from (2a) and (4), we deduce that:

$$\lim_{\substack{t \rightarrow t_{i+1} \\ t < t_{i+1}}} v(t) = -v_i \tag{5}$$

It follows that:

$$v_{i+1} = 0.8v_i \tag{6}$$

From (4), (5) and (6), we deduce the total duration of all flights until the  $n^{\text{th}}$  bounce, for any  $n \geq 0$ :

$$\begin{aligned} t_n - t_0 &= \sum_{i=0}^{n-1} t_{i+1} - t_i = \frac{2}{g} \sum_{i=0}^{n-1} v_i = \frac{2v_0}{g} \sum_{i=0}^{n-1} 0.8^i \\ &= \frac{2v_0}{g} \cdot \frac{1 - 0.8^n}{1 - 0.8} \end{aligned} \tag{7}$$

Finally, we conclude from (7) that:

$$\lim_{n \rightarrow \infty} t_n - t_0 = \frac{10v_0}{g} = 1 \tag{8}$$

In other terms time *converges*, meaning that the lifespan of the model is finite! This unexpected result is a manifestation of the *Zeno effect* which prevents the model from moving forward in time past the convergence limit, called the *Zeno point*. As a consequence, simulating a physical model past its Zeno point—which solvers kindly accept to do, as illustrated on Figure 1—means that we are no longer executing the alleged semantics of our favourite modelling language: we are observing a *free and necessarily wrong interpretation* of our program by the simulation tool. This is extremely embarrassing since Zeno effects cannot be spotted during simulation and since many practical models contain *abstraction mechanisms* such as Modelica’s *reinit* operator, conditional equations, etc., whose composition is known to yield such issues: how to prove that we are not actually running a model out of its actual time domain?

### 2.3 Discussion of the example

Physical system theorists may argue that the issue with our bouncing ball model comes from the usage of inappropriate abstraction mechanisms. Indeed, application of a classical energetic approach (such as the standard Bond Graph for instance) instead of the more direct equation-based approach would have naturally lead to a time-diverging model: from this point of view, a bouncing ball model is no more than a damped *oscillator*, whose solution exhibits an exponential decay as time diverges.

However, in practice, the price to pay to benefit from the virtues of energetic modelling is often too

high in terms of model complexity. Applied to our example for instance, such an approach would force us to express contact with the ground in a more detailed fashion (typically, by means of modulated C and R elements in Bond Graph, which poses the additional problem of finding an adequate modulation constraint). Quite often, we do not *know enough* of the phenomena to be able to write meaningful high-fidelity equations.

Furthermore, simulation performance may suffer dramatically from excess modelling details. For instance, if we simulate our original bouncing ball model with a solver based on the Trapezoidal rule (which is an order 2 method) we see that it performs very well (this is not very surprising since the flight trajectory is a parabola). On the other hand, performance dramatically collapses when the same solver is given a detailed version of the bouncing ball model. Interestingly, a closer look at the performance profile reveals that most of the CPU time is spent in solving the contribution of additional details of the high-fidelity model, which correspond precisely to the phenomenon we wanted to abstract away! Experimenting with real-world models brings us to similar conclusions: abstraction mechanisms help focusing on important parts of models, making the result often simpler and more efficient (but error-prone).

So it seems that we have to live with “dirty” abstractions: does it mean that we also have to accept unsound semantics for the sake of performance and simplicity? Fortunately no, as shown in the next sections: remember that our conclusions follow the initial premise assuming a time model based on  $\mathbb{R}$ .

### 2.4 Non-standard approach

A closer look at (7), which actually represents the *work* performed by our bouncing ball model in the course of the simulation, reveals the profound cause of our problem: we are assuming that *any* countable family of joined, nonempty sub-intervals (our  $[t_i, t_{i+1})$ ) eventually partitions any simulation interval. Of course, this property does not hold: it would be equivalent to assuming that the sum of the terms of *any* sequence of positive reals would certainly diverge.

How could it be possible to define an operational semantics that would ensure time divergence even in presence of behavioural abstractions? Let us make the following experiment: We successively apply the fixed-step forward Euler method (see Listing 2) to our bouncing ball model with decreasing step sizes, as shown in Figure 2. We observe progressively better



approximations of the solution, since the problem is stable and since we can find a step size that ensures stability of the method itself. Notice that since we have chosen a fixed-step scheme, the process of simulation simply cannot exhibit Zeno effects: it terminates after at most  $\lceil \frac{t_{end}-t_0}{h} \rceil$  steps, where  $h$  is the step size.

So why not define step-by-step calculation of the *ideal* trajectories generated by models—which is what operational semantics of hybrid systems is all about—in terms of the standard forward Euler method? The reason is that whatever the step size we choose this method only gives *approximations* of the trajectories of interest. Furthermore, as small as a candidate step size would be, it would always be possible to forge models that would require a smaller one, for instance, linear models having eigenvalues large enough to make the solution unstable and divergent (the first slope on Figure 2 shows the behaviour of the method in such a situation).

Notice also that this reasoning applies to *any* numerical integration method, not only forward Euler. The point is: numerical methods have to perform steps and there is no smallest possible step that would fit *all* models.

In order to define our reference calculation steps, we would have to choose a step that would be smaller than any positive real number. Furthermore, this reference step would have to be *positive* to ensure time divergence. This is precisely the definition of a *positive infinitesimal*, as these have been used in 17th and 18th centuries by mathematicians and physicists such as Leibniz and Newton.

The common idea of real numbers was different from the modern one. For instance, to compute the derivative of a given function  $f(x)$ , one would consider the *increment* of this function given an infinitesimal increment  $dx$  to  $x$ . Thus the derivative  $f'(x)$  was defined by setting

$$f'(x) = \frac{f(x+dx) - f(x)}{dx}.$$

For example, applying this reasoning to  $f(x) = x^2$ , one obtains the following computation

$$f'(x) = \frac{(x+dx)^2 - x^2}{dx} = \frac{2x dx + dx^2}{dx} = 2x + dx \approx 2x,$$

where the last relation signifies that  $dx$ , being infinitesimal, *vanishes* in the final expression.

The notion of infinitesimal was formalised in 1960s by Robinson (see [15]), by defining a set  ${}^*\mathbb{R}$  of *non-standard reals*, which is an ordered field extension of

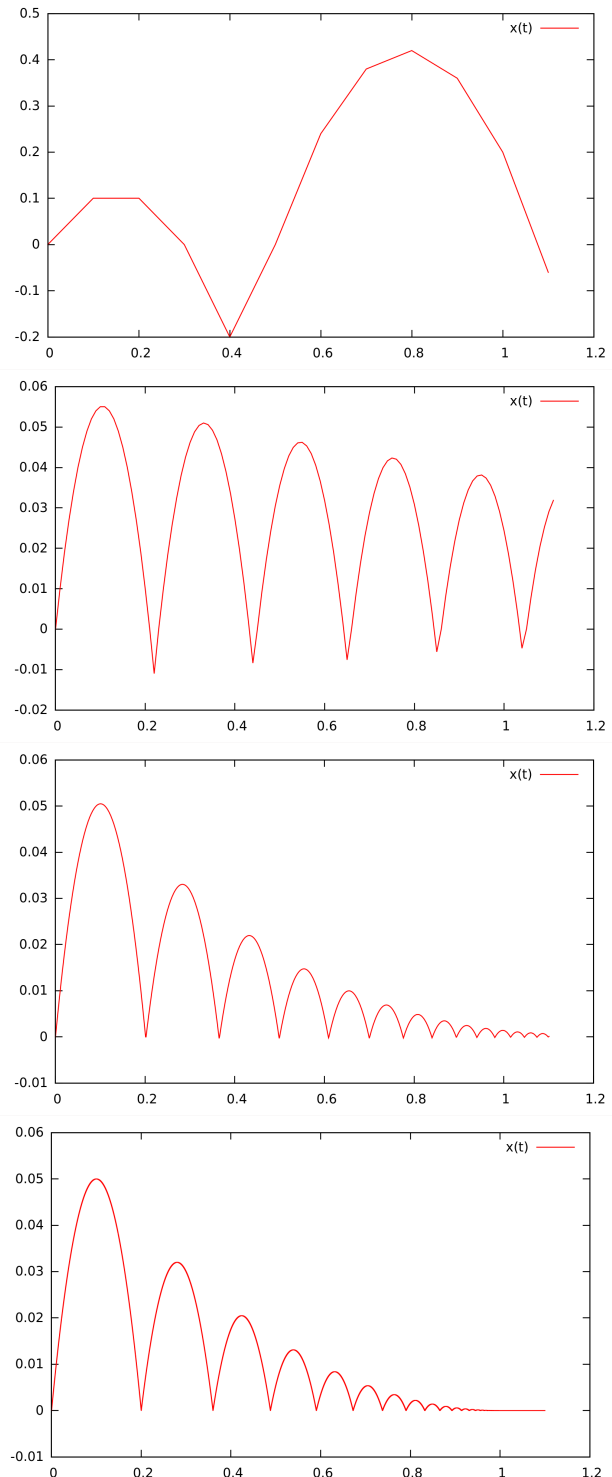


Figure 2: Simulation results for the bouncing ball model using forward Euler on  $[0, 1.1]$  with step sizes 0.1, 0.01,  $10^{-3}$  and  $10^{-6}$  respectively (altitude of the ball).

$\mathbb{R}$  that contains all usual real numbers, but also the *infinitesimal* reals and their inverses, which are the *infinitely great* reals, i.e. those with an absolute value strictly greater than any usual real number  $r \in \mathbb{R}$ . One also speaks of *finite* (or *limited*) reals  $x$ , such that  $|x| < y$  for *some* positive  $y \in \mathbb{R}$ ;

In particular, two non-standard real numbers  $x$  and  $y$  are said to be *infinitely close* (denoted by  $x \approx y$ ) if and only if  $x - y$  is infinitesimal.

Among all non-standard real numbers, one can, of course, consider the set  ${}^*\mathbb{Z}$  of *non-standard integers* that, on top of standard integers, contains infinitely great ones, having absolute value greater than any  $n \in \mathbb{N}$ . Similarly, it is possible to define the set of non-standard natural numbers  ${}^*\mathbb{N}_0$  (including zero).

The field  ${}^*\mathbb{R}$  of non-standard reals has been used by several authors to define operational semantics of continuous and hybrid systems [1, 2, 12, 16]. The key idea is to *change the definition of time*, by replacing standard reals with a subset of *non-standard* ones.

The proposal in [2, 16] consists in defining time as:

$$\mathbb{T} \stackrel{def}{=} \{ \varepsilon \cdot n \mid n \in {}^*\mathbb{N}_0 \} \quad (9)$$

for *some* reference positive infinitesimal  $\varepsilon$  (notice that *any* positive infinitesimal fits our constraints). It should be noted that  $\mathbb{T}$  contains arbitrary large numbers: in particular, it is possible, given any positive real  $t$ , to find  $n \in {}^*\mathbb{N}_0$  such that  $\varepsilon \cdot n$  is greater than  $t$ . This property is due to  ${}^*\mathbb{R}$  being Archimedean in the non-standard sense<sup>2</sup>, as shown in [2]. Also, it should be noted that although  $\mathbb{T}$  may contain absolutely no standard real (for some “unfortunate” choice of  $\varepsilon$ ), it is possible to *approximate* any standard real  $x$  by a non-standard real of the form  $\varepsilon \cdot n$  such that:

$$|x - \varepsilon \cdot n| < \varepsilon$$

which means that the error committed in the approximation is *negligible* in comparison to any positive element of  $\mathbb{R}$  (recall that  $\varepsilon$  is a positive *infinitesimal*).

We are now ready to define the meaning of a differential equation:

$$\dot{x} = f(x, t) \stackrel{def}{=} x_{next} = x + \varepsilon \cdot f(x, t) \quad (10)$$

where:

- $\varepsilon$  is the reference time step introduced in (9);
- $x_{next}$  is the “next” value of  $x$ , that is  $x(t + \varepsilon)$ .

<sup>2</sup>It can be shown, however, that  ${}^*\mathbb{R}$  is *not* Archimedean in the standard sense, because of the existence of infinite elements.

Listing 2: A Haskell program implementing the forward Euler method to the bouncing ball model.

```
euler t0 tEnd (v0, x0) h = step 0 (v0, x0)
  where
    step i (vNow, xNow)
      | tNow > tEnd = []
      | otherwise =
          (tNow, xNow) : step (i + 1) (vNext, xNext)
    where
      tNow          = t0 + h * fromInteger i
      (vNext, xNext) = advance tNow (vNow, xNow)
    advance _ (vNow, xNow)
      | xNow < 0.0 = (-0.8 * vNow, 0.0)
      | otherwise = (vNow - h * 10.0, xNow + h * vNow)
```

Why would an operational semantics based on (9) and (10) prevent Zeno effects? For exactly the same reason why standard forward Euler method prevents them: because time is forced to advance by fixed—although infinitesimal—steps and because the non-standard Archimedean property of  ${}^*\mathbb{R}$  allows arbitrarily large times to be overstepped.

However, we are not completely done yet. Indeed, due to the multiplication by  $\varepsilon$  in (10), our operational semantics maps time instants as well as values of real signals to *non-standard* reals, although we would like our models to eventually yield standard real values. Fortunately, any limited non-standard real can be *uniquely* represented as the sum of their standard and infinitesimal parts [2, 12, 16]. All we need to do is to eventually discard infinitesimal parts of non-standard reals in order to construct our final standard signals.

Operational semantics based on this idea lead to *uniform* treatment of discrete and continuous dynamics: differential equations, reset equations (e.g. Modelica’s *reinit* operator) and difference equations are actually all treated as non-standard difference equations.<sup>3</sup> As an illustration of this, the program of Listing 2 implements the operational semantics of our bouncing ball model, provided

- $v0$  and  $x0$  are standard reals and  $h$  is an arbitrary infinitesimal;
- integers (used as step indexes) are non-standard;
- infinitesimal parts of the output are discarded.

Notice that, past  $t = 1$ , as experiments with standard Euler method suggest, the ball sticks to the ground—more exactly, standardisation of the model’s variables

<sup>3</sup>It is interesting to contrast this uniformity with Modelica’s informal semantics as given in the current language specification.

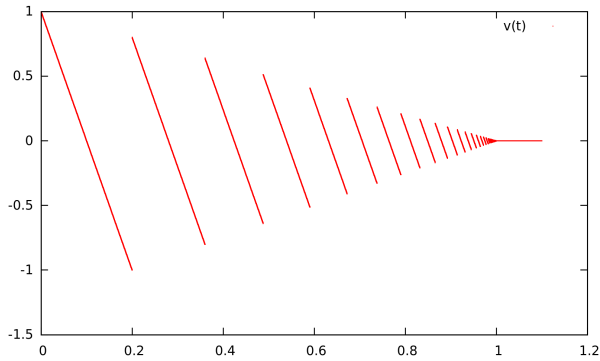


Figure 3: Velocity of the ball on  $[0, 1.1]$  resulting from interpretation of the bouncing ball model with non-standard semantics.

yields two standard functions of time which map every  $t \geq 1$  with zero. Indeed, from (6) we know that the standard part of  $v$  converges to zero as  $t$  converges to 1. It follows that for every  $t \geq 1$ , dynamic equations of the model, as a consequence of (10), only produce infinitesimal values since the product of  $\varepsilon$  by any limited real yields an infinitesimal number.

### 3 Behavioural abstraction issues

As shown in [1, 2], operational semantics built on time model (15) successfully explain the behaviour of programs like the bouncing ball model of Listing 1—even beyond  $t = 1$ . In particular, abstraction of the bounce phenomenon meets our expectations: velocity and altitude *instantaneously* take an explicitly specified value at bounce time and then *continuously* evolve from this new starting point until the next bounce.

It should be noted that, in this model, while altitude keeps its continuous character around bounce instants, velocity *loses* it, as illustrated in Figure 3. Nevertheless, interpretation of the bouncing ball model remains satisfactory with respect to the requirements which only demand a high-fidelity behaviour between bounces and a correct (and instantaneous) repositioning of velocity and altitude at bounce time for the next flight to start.

However, as shown below, loss of continuity due to abstraction may cause practical models of systems to fail unexpectedly.

#### 3.1 A problematic example

We consider in Listing 3 a simple fuse sub-model, which behaves like an electrical switch that is closed by default but that can eventually become open if the branch current exceeds a limit.

Listing 3: A fuse sub-model, in Modelica.

```
import Modelica.Electrical.Analog.*;

model Fuse
  extends Interfaces.OnePort;
  parameter Real iMax;
  parameter Real Ron;
  parameter Real Roff;
  protected Real R;
  protected Boolean on;
  initial equation
    on = true;
  equation
    when i > iMax then
      on = false;
    end when;
    R = if on then Ron else Roff;
    v = R * i;
end Fuse;
```

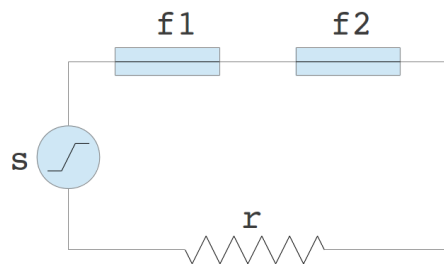


Figure 4: An electrical circuit using two instances of the fuse sub-model defined in Listing 3.

Listing 4: A ramp voltage source sub-model, in Mod-  
 elica.

```

import Modelica.Electrical.Analog.*;

model RampVoltageSource
  extends Interfaces.OnePort;
  parameter Real startTime;
  parameter Real k;
  parameter Real vMax;
  equation
    v =
      if time >= startTime then
        min(k * (time - startTime), vMax)
      else 0.0;
  end RampVoltageSource;
    
```

Using this sub-model, we build a model of the system shown in Figure 4, which is composed of a ramp voltage source (defined in Listing 4), two fuses and a simple linear resistor, in series. We suppose in this model that the fuses have *distinct* rated currents. Notice that, according to the definition of the fuse sub-model, the time required by a fuse to break the circuit is negligible with respect to the time required by the ramp voltage source to reach its maximum value<sup>4</sup>.

Suppose we are given these parameter bindings:

$$\text{src.startTime} = 0.1 \quad (11a)$$

$$\text{src.k} = 2 \quad (11b)$$

$$\text{src.vMax} = 1 \quad (11c)$$

$$\text{f1.iMax} = 0.005 \quad (11d)$$

$$\text{f1.Ron} = 10^{-6} \quad (11e)$$

$$\text{f1.Roff} = 10^6 \quad (11f)$$

$$\text{f2.iMax} = 0.006 \quad (11g)$$

$$\text{f2.Ron} = 10^{-6} \quad (11h)$$

$$\text{f2.Roff} = 10^6 \quad (11i)$$

$$\text{r.R} = 100 \quad (11j)$$

and this initial state:

$$\text{time} = 0 \quad (12a)$$

$$\text{f1.on} = \text{T} \quad (12b)$$

$$\text{f2.on} = \text{T} \quad (12c)$$

According to our non-standard semantics, we have to solve the following dynamic equations to determine

<sup>4</sup>This property is enforced by the use of a *when* clause in the definition of the fuse sub-model in Listing 3.

the dynamic behaviour of our model:

$$\text{src.v} = \begin{cases} \min(2(\text{time} - 0.1), 1) & \text{if } \text{time} \geq 0.1 \\ 0 & \text{otherwise} \end{cases} \quad (13a)$$

$$\text{src.i} = i \quad (13b)$$

$$\text{f1.R} = \begin{cases} 10^{-6} & \text{if } \text{f1.on} \\ 10^6 & \text{otherwise} \end{cases} \quad (13c)$$

$$\text{f1.v} = \text{f1.R} \cdot i \quad (13d)$$

$$\text{f1.i} = i \quad (13e)$$

$$\text{f2.R} = \begin{cases} 10^{-6} & \text{if } \text{f2.on} \\ 10^6 & \text{otherwise} \end{cases} \quad (13f)$$

$$\text{f2.v} = \text{f2.R} \cdot i \quad (13g)$$

$$\text{f2.i} = i \quad (13h)$$

$$\text{r.v} = 100i \quad (13i)$$

$$\text{r.i} = i \quad (13j)$$

$$\text{time}_{\text{next}} = \text{time} + \varepsilon \quad (13k)$$

$$\text{f1.on}_{\text{next}} = \text{f1.on} \wedge \neg(i > 0.005) \quad (13l)$$

$$\text{f2.on}_{\text{next}} = \text{f2.on} \wedge \neg(i > 0.006) \quad (13m)$$

where

$$i = \frac{\text{src.v}}{100 + \text{f1.R} + \text{f2.R}} \quad (13n)$$

Notice that this model has *three* state variables, namely *time*, *f1.on* and *f2.on*, hence the three non-standard difference equations (13k), (13l) and (13m).

Figure 5 shows the corresponding results. Notice that, as expected, only the first fuse melts (see second slope in Figure 5) since its rated current is lower than that of the first fuse (see (11d) and (11g)).

In this first experiment, only the behaviour of fuses has been abstracted away by considering their melt duration to be negligible with respect to the raise duration of the ramp source. But what happens if we also abstract the behaviour of the ramp source, considering its raise duration negligible with respect to the entire operating duration of the system?

Abstracting the ramp source in this context means replacing it with a *step* source, leading to a new system of dynamic equations, where (13a) is replaced with:

$$\text{src.v} = \begin{cases} 1 & \text{if } \text{time} > 0.1 \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

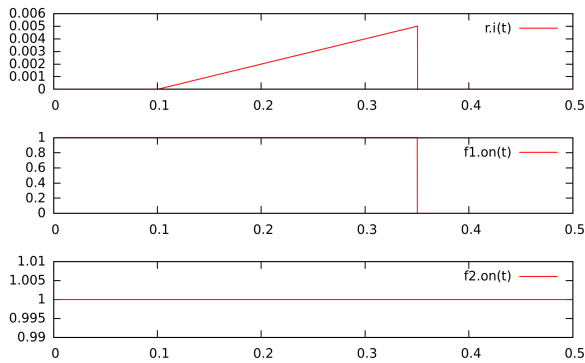


Figure 5: Solution of (13) with parameter bindings (11) and initial state (12).

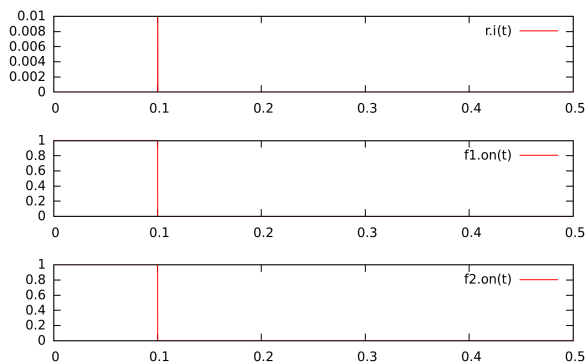


Figure 6: Solution of the system of dynamic equations resulting from the abstraction of the ramp voltage source.

Figure 6 shows the corresponding results. We can see that *both* fuses melt as a consequence of the raise of the voltage source signal. This fact contradicts our initial assumption of a model where only one fuse may melt, if ever. This last experiment has shown that an operational semantics based on a global, fixed infinitesimal step does not preserve *strict ordering* of events when behavioural abstractions are composed.

### 3.2 Conclusions from the experiments

The reason why we observe an accidental synchronisation of events after abstracting the behaviour of the ramp voltage source is that the time-line is “running out of available free slots” where to insert possible intermediate events. In our example, events corresponding to melts, whenever they eventually occur, must necessarily do so *strictly before* the ramp signal of the voltage source reaches its maximum value.<sup>5</sup> However, in a fixed-step non-standard operational semantics, when a standard difference equation is activated

<sup>5</sup>Because it is the voltage raise itself, after its “conversion” to current through the load  $r$ , that triggers an eventual melt.

to reset the value of a state variable, or when a conditional equation switches (as in our example), new values are scheduled for the *next* instant, leaving no room for possible intermediate events to occur during this transition. So it seems that we should *densify* time in the neighbourhood of “urgent” actions.

## 4 Proposed semantic model

In this section, we introduce some basic notions that will be used to construct our model of a signal as well as defining the meaning of a differential equation.

We suppose given a positive infinitesimal real  $\varepsilon$  that we call the *real activity threshold*. Notice that, contrary to traditional fixed-step simulation, the semantics of the system is independent from the specific choice of  $\varepsilon$ .

For  $r \in {}^*\mathbb{R}$ , we denote  $r + \varepsilon \cdot {}^*\mathbb{Z} \stackrel{def}{=} \{r + \varepsilon \cdot k \mid k \in {}^*\mathbb{Z}\}$ , where  ${}^*\mathbb{Z}$  is the set of non-standard integers. We define the base time-line  ${}^*\mathbb{T}$  as the non-negative values of  ${}^*\mathbb{R}$ :

$${}^*\mathbb{T} \stackrel{def}{=} {}^*\mathbb{R}_0^+ \quad (15)$$

Thus,  ${}^*\mathbb{T}$  is a *linear continuum* (under the standard temporal order  $<$ ), which contains all non-negative standard real numbers. An *instant* is an element of  ${}^*\mathbb{T}$ . In particular, we have non-standard instants.

Notice that, contrary to [1, 2, 16], we propose here a base time-line that is neither discrete, nor well-ordered. However, since  ${}^*\mathbb{T}$  is *densely ordered*, intermediate instants exist between any two given distinct instants.

The question that arises now is: how to recover discreteness (as required to express the notion of *next* instant) as well as time divergence? The key idea lies in a *generalisation* of the usual concept of *clock* as encountered in synchronous languages.

### 4.1 Time, signals and dates

#### 4.1.1 Time

A *time signal*  $t$  with the initial value  $t_0 \in {}^*\mathbb{R}$  is defined as the following right-continuous step map:

$$\begin{aligned} t : {}^*\mathbb{T} &\rightarrow t_0 + \varepsilon \cdot {}^*\mathbb{Z} \\ \tau &\mapsto t_0 + \varepsilon \cdot n_\tau \end{aligned} \quad (16)$$

with

$$n_\tau = \left\lfloor \frac{\tau}{\varepsilon} \right\rfloor \in {}^*\mathbb{N}_0$$

where  $\lfloor \cdot \rfloor$  denotes the non-standard floor function.

In particular, for any  $\tau \in \mathbb{R}_0^+$  we have:

$$st(t(\tau) - t_0) = \tau$$

meaning that the standardisation of  $t$  coincides with a standard time signal that would start from  $st(t_0)$ .

By definition, a *date* is an element of the co-domain of a time signal.

Notice the fundamental change of perspective with respect to [1, 2, 16]: instead of discretising the *domain* of the time signal, we discretise its *co-domain*. Thus, we can associate a discrete and well-ordered set of dates with a time signal, despite its densely ordered domain. Notice also that dates and instants are different concepts: while instants are densely ordered, dates are not. In the next sections, we show how to extend this idea to *any* signal.

### 4.1.2 Signals

We call a (*non-standard*) signal  $x$  a map from a subset of  ${}^*\mathbb{T}$ —the *domain* of the signal, denoted  $\text{Dom}x$ —to a *discrete* set—its *co-domain*, denoted  $\text{Codom}x$ . A signal maps instants to *values*.

We distinguish two families of signals:

**discrete-time signals** have their domain (called their *clock*) in a discrete subset of  ${}^*\mathbb{T}$ ;

**dense-time signals** are right-continuous step signals, whereof the domain is  ${}^*\mathbb{T}$ .

A few remarks can be made regarding this taxonomy:

- We do not require zero (i.e. the least element of  ${}^*\mathbb{T}$ ) to belong to the domain of a discrete-time signal: zero is only guaranteed to be a lower bound of the domain. In other terms, the birth instant of a discrete-time signal does not necessarily coincide with the birth instant of the model.
- Physical signals introduced in [1, 2] belong to the *discrete-time* family of signals.
- In some sense, compared with [2], we loose some uniformity by introducing two kinds of signal domains, with distinct topologies. However, practical implementations of modelling languages have to make a semantic distinction between signals at some point (to avoid ill-posed problems). In [1], for instance, the type system classifies signals in “discrete” and “continuous” ones (although they share the same representation).

Notice that, according to the definition of a signal, the co-domain of any signal is discrete. This raises the question of the representation of *real* signals in general, and *continuous* (or *physical*) signals in particular. We impose the following restrictions:

**real signals** have co-domains of the form  $r + \varepsilon \cdot {}^*\mathbb{Z}$ , where  $\varepsilon$  is the real activity threshold and  $r$  the start value of the signal;

**continuous (or physical) signals** are dense-time real signals that can only change their value by  $\pm\varepsilon$ .

### 4.1.3 Signal activity

The *activity* of a signal  $x$ , denoted  $\text{Act}x$ , is the discrete (possibly finite, but often non enumerable) subset of  $\text{Dom}x$  defined as:

$$\text{Act}x = \{\tau \mid x(\tau^-) \neq x(\tau)\} \quad (17)$$

where

$$x(\tau^-) = \lim_{\substack{\tau' \rightarrow \tau \\ \tau' < \tau}} x(\tau')$$

Intuitively, the activity of a signal can be seen as the set of instants corresponding to its “perceptible changes” from the point of view of an external observer. Notice that we have required the co-domain of signals to be discrete: this implies that the activity of a signal  $x$  contains *all* the instants at which “something happened” to  $x$  from an external observer point of view, whatever the accuracy of the measure.

## 4.2 Differential equations

If one assumes the existence of a *maximal* clock such as

$$\{\varepsilon \cdot n \mid n \in {}^*\mathbb{N}_0\}$$

then the sole concept of non-standard *difference equation* suffices to express dynamic behaviour of models. Indeed, as explained in previous sections, the solution yielded by the (non-standard) forward Euler scheme coincides with the actual, standard solution of the corresponding differential equation, after standardisation. But we have also seen that this uniform approach reaches its limits when composition of abstract models comes into play.

Density of possibly detectable events should be ideally correlated to the behaviour of signals instead of being imposed by a rigid, fixed-step scheme: indeed, more *activity* potentially implies more events.

Also, if one would be able to *predict* the occurrence of an event from the behaviour of the implied

signal(s) instead of waiting for the *next* time instant and then realise that *several* pending events need to be (wrongly) treated simultaneously, one would avoid the issues mentioned in previous sections.

The key idea is to define differential equations in such a way that activity of signals can be predicted, and then to use the concept of activity instead of the concept of clock to represent (mutually desynchronised) event sources. This idea is not new: B. P. Zeigler [17] already introduced most of the necessary concepts in his theory of systems. QSS solvers [5] are practical applications of this theory.

Recall that in [1, 2], the semantics of a differential equation was defined by means of a non-standard version of the forward Euler method, which is the simplest order-one method based on the classical time slicing approach. In the following, we will define the meaning of a differential equation by means of a non-standard order-one QSS-like method and show how this new approach solves behavioural abstraction issues.

Assume that the following are given:

- a non-empty interval  $[\tau_a, \tau_b) \subseteq {}^*\mathbb{T}$ ,
- a non-standard real number  $x_0 \in {}^*\mathbb{R}$ ,
- a non-standard real signal  $y : [\tau_a, \tau_b) \rightarrow y_0 + \varepsilon \cdot {}^*\mathbb{Z}$ .

Then the differential equation

$$\text{der}(x, x_0) := y \quad (18)$$

defines  $x$  on  $[\tau_a, \tau_b)$  as follows.

Let  $(w_i)_{i \in {}^*\mathbb{N}_0}$  be a family of  ${}^*\mathbb{R}$ -valued maps defined as:

$$w_0(\tau) = x_0 + (\tau - \tau_0) \cdot y(\tau_0) \quad (19a)$$

$$w_{n+1}(\tau) = w_n(\tau_{n+1}) + (\tau - \tau_{n+1}) \cdot y(\tau_{n+1}) \quad (19b)$$

where

$$\tau_0 = \tau_a \quad (20a)$$

$$\tau_{n+1} = \text{Inf}_{[\tau_a, \tau_b)} \left( \{ \tau \mid \tau > \tau_n \wedge \text{event}_n(\tau) \} \cup \{ \tau_b \} \right) \quad (20b)$$

and

$$\begin{aligned} \text{event}_n(\tau) = & \\ & (\tau \in \text{Act}y) \vee \\ & \left( (w_n(\tau) \in x_0 + \varepsilon \cdot {}^*\mathbb{Z}) \wedge (w_n(\tau) \neq w_n(\tau_n)) \right) \end{aligned} \quad (21)$$

Defining the co-product

$$w = \bigoplus_{i \in {}^*\mathbb{N}_0} w_i \Big|_{[\tau_i, \tau_{i+1})} \quad (22)$$

then, finally:

$$\begin{aligned} x : [\tau_a, \tau_b) &\rightarrow x_0 + \varepsilon \cdot {}^*\mathbb{Z} \\ \tau &\mapsto w(\text{last}(\tau)) \end{aligned} \quad (23)$$

where

$$\begin{aligned} \text{last}(\tau) = & \\ \text{Sup}_{[\tau_a, \tau_b)} \{ \tau' \mid \tau' \leq \tau \wedge w(\tau') \in x_0 + \varepsilon \cdot {}^*\mathbb{Z} \} & \end{aligned} \quad (24)$$

The idea is the following:

- $w$  represents the “private” behaviour of the defined signal  $x$ : starting from  $x_0$  at  $\tau_a$ , it evolves linearly between two consecutive event instants ((19a) and (19b)) until  $\tau_b$ ;
  - as stated by (21), events affecting the behaviour of  $w$  have two possible causes:
    - $y$  (defining the right-hand side of the differential equation) has evolved perceptibly, i.e. its value has drifted by  $\pm\varepsilon$ , since the previous event (including initialisation),
    - $w$  itself has evolved perceptibly since the previous event (including initialisation);
  - each time an event affecting  $w$  occurs, its gradient is reevaluated, then  $w$  evolves linearly until the next event;
  - the “public” behaviour of  $x$  (i.e. as seen by an external observer) is a piece-wise constant approximation of its “private” behaviour  $w$  such that both maps coincide at event instants corresponding to perceptible changes of  $w$ , as stated by (24).
- Figure 7 illustrates the process (the  $w_i$  appear as black linear slopes, and  $x$  is the blue slope):
- blue bullets indicate events resulting from the evolution of  $w$  itself;
  - red bullets indicate events resulting from the evolution of  $y$  (notice that they are not necessarily “synchronised” with  $\varepsilon$ -steps);
  - green bullets indicate “skipped targets” (i.e. events that would have resulted from the sole evolution of  $w$  but that have been “intercepted” by an event caused by  $y$ ).

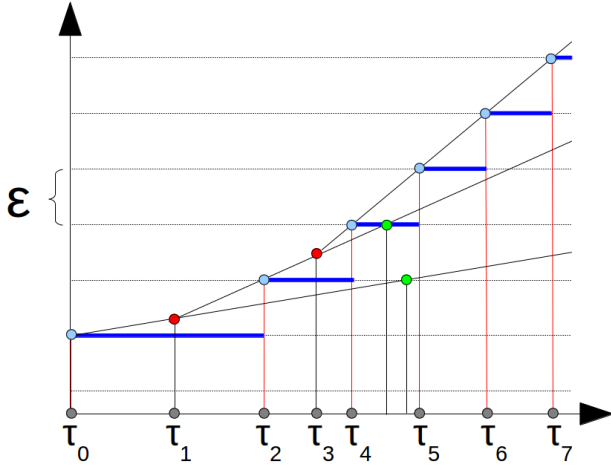


Figure 7: Semantics of a differential equation.

Notice that we have defined the semantics of a differential equation on an interval of the form  $[\tau_a, \tau_b)$  instead of defining it on the whole time-line  ${}^*\mathbb{T}$ . The reason is that we want differential equations to be defined modularly, as required in particular by behavioural abstraction: it should be possible to define signals where ideal phases alternate with high-fidelity ones.

Clearly, the set of continuous signals is closed under the application of continuous functions. Therefore any family of signals defined solely by means of differential equations and by algebraic constraints like

$$y = f(x_1, \dots, x_n) \quad (25)$$

only contains continuous signals. Regarding (25) we have in particular:

$$\text{Act}y \subseteq \text{Act}x_1 \cup \dots \cup \text{Act}x_n \quad (26)$$

meaning that such an algebraic constraint acts as an “activity filter”.

### 4.3 Behavioural abstraction support

Below, we suppose given an additional positive infinitesimal constant  $\delta$ .

We model a reset equation constraining a continuous signal  $x$  on  $[\tau_a, \tau_a + \delta)$  by means of the following differential equation:

$$\text{der}(x, x_0) := \frac{x_1 - x_0}{\delta} \quad (27)$$

where  $x_0$  is the value of  $x$  at  $\tau_a$ , and  $x_1$  is the “target value” to be reached by  $x$  at  $\tau_a + \delta$  as a result of the reset operation.<sup>6</sup>

<sup>6</sup>In Modelica, this is given by the second argument of `reinit` (see Listing 1 for an example).

Notice that the right-hand side of (27) is constant and infinite. By the definition of a differential equation given in Section 4.2,  $x$  reaches  $x_1$  (more precisely: the first element of  $x_0 + \varepsilon \cdot {}^*\mathbb{Z}$  that is greater than or equal to  $x_1$ ) after  $\delta$  units of time (i.e. the length of  $[\tau_a, \tau_a + \delta)$ ). Notice that this definition of a reset equation involves a *duration* (i.e. the time spent by the signal to reach its target value). Since this duration is infinitesimal, we actually observe the desired behaviour: the reset is infinitely fast compared to any standard phenomenon.

Recall, however, that we want to support proper composition of such abstraction mechanisms. In order to refine the above definition to achieve this goal, we first associate with each continuous signal defined in the model an *abstraction level*  $n \in \mathbb{N}_0$ <sup>7</sup> which represents the maximal number of nested reset equations that might contribute to the definition of the signal:

- a independent signal that is not reset in the model has the abstraction level 0;
- a signal that is reset in the model by equations involving only signals whose abstraction level is at most  $n - 1$  has abstraction level  $n$ ;
- a signal that is defined by an algebraic constraint inherits the highest abstraction level among those of the signals appearing in its definition.

Notice that, in a language implementing our semantic model, the abstraction level defined as above can always be statically computed by elementary data-flow analysis for an arbitrary model without circular dependencies between reset equations.

We are now in the position to refine (27) as follows. A signal  $x$  with abstraction level  $n$  and the current value  $x_0$  is reset to  $x_1$  at  $\tau_a$  by means of the following differential equation activated on  $[\tau_a, \tau_a + \delta^n)$ :

$$\text{der}(x, x_0) := \frac{x_1 - x_0}{\delta^n} \quad (28)$$

The idea is that signals with higher abstraction levels should reset infinitely faster than others: by taking successive powers of  $\delta$ , we achieve precisely this effect.

What about the Zeno effect under behavioural abstraction in such a semantic model?

Notice that we can assume that continuous signals do not diverge in value during integration (otherwise the model is considered singular) and let the highest level of abstraction in a given model be  $n$ . By (28), this

<sup>7</sup>A standard natural number.



means that in this model, the fastest reset terminates in  $\delta^n$  units of time. Hence, integration of each signal terminates in at most  $\lceil (\tau_{end} - \tau_0) / \delta^n \rceil$  steps, where  $\tau_0$  and  $\tau_{end}$  denote respectively the start time and the end time of the integration. Thus, provided that the maximum abstraction level in the model is finite, continuous signals always diverge in time.

#### 4.4 Application Example

Let us consider again the model of Figure 4, where we suppose that the behaviour of the voltage source and of the fuses have been abstracted. How does our proposal fare in such a case?

Before we answer this question, we need to examine the case of conditional equations.

Conditional equations are used to express “mode changes” in physical models, typically leading to equation *switching*, as in our electrical model example. As a consequence, conditional equations potentially lead to the exact same issues encountered with reset equations. Same symptoms, same medication: we just need to find a way to introduce the concept of infinitely fast transition in the semantics of conditional equations. This is achieved as follows.

Suppose that  $c : *T \rightarrow \{F, T\}$  is a boolean signal whose value is F until the event instant  $\tau_0$ , after which it takes the value T. Suppose also that  $x : *T \rightarrow x_0 + \varepsilon \cdot *Z$  and  $y : *T \rightarrow y_0 + \varepsilon \cdot *Z$  are two given physical signals. Then the semantics of a conditional real function can be defined as:

$$\text{if } c \text{ then } x \text{ else } y \stackrel{def}{=} \lambda \cdot x + (1 - \lambda) \cdot y \quad (29)$$

where

$$\lambda = \begin{cases} 0 & \text{on } \{\tau \mid \tau \leq \tau_0\} \\ \frac{\tau - \tau_0}{\delta} & \text{on } \{\tau \mid \tau_0 < \tau \leq \tau_0 + \delta\} \\ 1 & \text{on } \{\tau \mid \tau > \tau_0 + \delta\} \end{cases} \quad (30)$$

It can easily be shown that signals defined by conditional real functions are continuous signals.

Coming back to our example, what happens when the voltage source makes a step? The voltage  $\text{src.v}$  gradually evolves from its start value 0 to its maximum value 1. During the rise, it traverses the surface corresponding to

$$\text{src.v} = f1.iMax \cdot (r.R + f1.R + f2.R)$$

because, given our parameter settings and the values of the internal resistances of both fuses before the first melt, we have:

$$f1.iMax \cdot (r.R + f1.R + f2.R) = 0.500000001$$

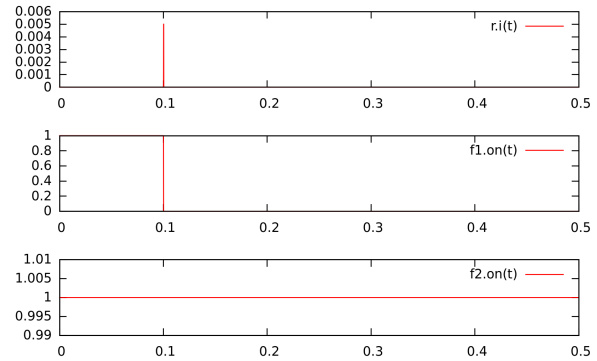


Figure 8: Solution of the system of dynamic equations resulting from the abstraction of the ramp voltage source.

which belongs to  $[0, 1]$ .

But, according to (13n), this is precisely the voltage required to induce a current equal to the rated current of f1 in the circuit. Consequently, f1 melts, but not f2. Figure 8 illustrates the result of interpreting the model with our semantics.

## 5 Conclusion and Future Work

In this paper, we have extended the pioneer work of [2, 16] by proposing a non-standard operational semantics supporting compositional behavioural abstraction. As demonstrated by [1], non-standard semantics can be used to give rigorous interpretation of hybrid modelling languages such as Modelica.

The most obvious practical application of our work would certainly be the design and development of a simulator that would conform to our semantic model: one of us (S. Furic) is currently working in this direction in the course of the French funded project AGéSys.

## 6 Acknowledgements

We are indebted to Albert Benveniste, Benoît Caillaud, Marc Pouzet and Timothy Bourke for having launched the topic: their impressive work and their availability for exchanging on modelling language semantic topics has greatly helped us in this work. Many thanks also to Ramine Nikoukhah, for his help in understanding hybrid modelling issues and for having brought to our attention the central role of some synchronous language concepts, especially those of the SIGNAL language. Many thanks also to Belgacem Ben Hedia and to Étienne Hamelin, from CEA for the many interesting discussions. This paper results in large parts from the

fruitful collaboration between CEA and LMS Imagine in the course of the ITEA2 project OpenProd.

## References

- [1] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78:877–910, May 2012. doi: 10.1016/j.jcss.2011.08.009.
- [2] Simon Bliudze and Daniel Krob. Modelling of complex systems: Systems as dataflow machines. *Fundamenta Informaticae*, 91:1–24, 2009. doi: 10.3233/FI-2009-0001.
- [3] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with ODEs. In *16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, pages 113–118, Philadelphia, USA, March 2013.
- [4] Stephen L Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer, 2010. ISBN 978-1-4419-5527-2.
- [5] François E. Cellier, Ernesto Kofman, Gustavo Migoni, and Mario Bortolotto. Quantized state system simulation. *Proceedings of Grand Challenges in Modeling and Simulation (GCMS'08)*, pages 504–510, 2008.
- [6] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.
- [7] Goran Frehse, Colas Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-22110-1\_30.
- [8] Peter Fritzson. *Introduction to modeling and simulation of technical and physical systems with Modelica*. Wiley-IEEE Press, 2011. ISBN 978-1-118-01068-6.
- [9] Sébastien Furic. Enforcing model composability in Modelica. In *Proceedings of the 7th International Modelica Conference, Como, Italy*, pages 868–879, 2009.
- [10] Boris Golden, Marc Aiguier, and Daniel Krob. Modeling of complex systems II: A minimalist and unified semantics for heterogeneous integrated systems. *Applied Mathematics and Computation*, 218(16):8039–8055, 2012.
- [11] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96*, pages 278–292. IEEE Society Press, 1996.
- [12] H. Jerome Keisler. *Foundations of Infinitesimal Calculus*. On-line Edition, 2007. URL <http://www.math.wisc.edu/~keisler/foundations.html>.
- [13] Michal Konečný, Walid Taha, Jan Duracz, Adam Duracz, and Aaron Ames. Enclosing the behavior of a hybrid system up to and beyond a Zeno point. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*, pages 120–125, 2013. doi: 10.1109/CPSNA.2013.6614258.
- [14] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 25–53. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25108-8. doi: 10.1007/978-3-540-31954-2\_2.
- [15] Abraham Robinson. *Non Standard Analysis*. North Holland, 1966.
- [16] Heinrich Rust. *Operational Semantics for Timed Systems: A Non-standard Approach to Uniform Modeling of Timed and Hybrid Systems*, volume 3456 of *Lecture Notes in Computer Science*. Springer, 2005. ISBN 3-540-25576-1. doi: 10.1007/978-3-540-32008-1.
- [17] Bernard P. Zeigler and Jong Sik Lee. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. In *SPIE Proceedings*, volume 3369, pages 49–58, 1998.

# An example of beneficial use of variable-structure modeling to enhance an existing rocket model

Alexandra Mehlhase Daniel Gomez Esperon Julien Bergmann Marcel Merkle  
Technical University of Berlin, Department of Software Engineering  
Ernst-Reuter-Platz 7, 10587 Berlin

## Abstract

This paper introduces a rocket model and discusses the advantages of refining it using a variable-structure approach to remodel critical parts. Both versions of the model are implemented in Modelica and were simulated using Dymola as simulation environment. The DySMo framework, which supports the simulation of variable-structure models in common simulation environments, was used to facilitate the redesign. The general benefits of the variable-structure approach are presented, and on the basis of the rocket model we present that simulation time and the data volume of the simulation can be reduced while maintaining the accuracy of the simulation results.

*variable-structure modeling, simulation speed, data reduction, reduce equation stiffness, rocket launch*

## 1 Introduction

In this paper, we regard the benefits of modeling a moon bound rocket using variable-structure to increase the simulation speed and to reduce the amount of saved simulation data.

The aim of our simulation is to predict the trajectory of the rocket, beginning with the ignition on earth's surface up until it reaches the moon as destination. The rocket is multi-staged, and as such consists of three booster modules and a payload module without means of propulsion. The model takes into account the chemical reactions in the boosters combustion chambers, which generate the thrust, the gravity of both earth and moon as well as atmospheric influences. First, we introduce a classical implementation of the model, which we will then compare to a redesigned version that uses the variable-structure approach.

The classical model calculates all components during the entire simulation. However the time frame

of the chemical reactions and of the rocket's movement are very different, which results in a stiff system of equations. This stiff equation system necessitates small step sizes, although a great part of the equation set would permit rather large time steps. Additionally, the chemical reactions only need to be calculated as long as the thrust of the rocket has not reached a steady state.

As a result, the simulation of the classical model generates an exceeding overhead of unnecessary simulation data and calculations. The ability to enable and disable equations during runtime, holding certain values constant for a given period of time, would allow to reduce this overhead. However, Modelica requires a constant set of equations and does not allow to change the equation system while simulating. All equations have to be solved during the whole simulation. In special cases and with the use of some workarounds the equation system can be manipulated to a certain degree, but not in the extend we want to use in this paper.

In section 2, we introduce the concept of variable-structure models and discuss different approaches to implement and simulate them. We will then give a detailed overview over the rocket model as well as the redesigned version in section 3. We compare the simulation results of both models, regarding performance and accuracy as well as the volume of generated data, in section 4.

## 2 Variable-Structure Modeling

The aim of variable-structure modeling is to improve models by introducing the means to change their equation and variable set during simulation. This is realized by encapsulating certain sets of equations and variables into different models, to which we refer as *modes*, and implementing a way to switch between them. Each transition is triggered by a predefined

switching condition. The mode switch is then realized by storing the end values of certain variables and using them to initialize variables of the next mode. The decision, which end values are used to initialize variables of the subsequent mode, has to be made by the modeler.

Variable-structure models are the topic of numerous publications, although they are often referred to by different names. In [6, 8] they are introduced as 'Multi-models', whereas they are referred to as 'Structurally dynamic systems' in [4] and 'Variable structure systems' in [10]. As they feature discrete mode switches as well as continuous equation systems they are often considered a variant of hybrid systems. A noteworthy method of formally specifying hybrid systems is the DEVS formalism, introduced in [9, 7].

Currently, altering the set of equations at runtime is not easily achieved using common Modelica simulation environments, which we want to use for the rocket simulation. However, several approaches to introduce variable-structure to a Modelica model have been devised, an overview is given in [1]. One way is presented in [2], where conditional statements are used to enable and disable certain equations based on disjunct conditions, which distinguish the current mode of the model. Such an approach is classified as *Maximal state space*, as the models state space is static and holds all states regardless of the current mode. This may lead to complicated mode switching procedures, as more than just the equations relevant at the current time have to be taken into account. Additionally, it may prove difficult to add new modes to an existing model. We followed another approach, termed *Hybrid decomposition*, where each mode is implemented as a separate model and the simulation switches between these models based on switching conditions. Mosilab [5] and SOL [10] are two approaches which enable the user to create variable-structure models. But since they are based on own languages it would be necessary to re-implement the rocket model in the specific language. In our approach we want to use common simulation environments. A framework to use common simulation environments is DySMo [3]. This framework is implemented in Python and allows a user to define a variable-structure model. The framework then handles the switches between the different modes automatically. In this framework Simulink, Dymola and OpenModelica are integrated. The communication with the tools is based on a communication interface integrated in Python. New interfaces can easily be added to the framework and then be used as simulation

environments for the variable-structure model.

A basic overview of the sequence used in the framework for switching between different modes is illustrated in Figure 1. Each of the gray boxes is a method defined in the communication interface from the specific simulation environment for the currently active mode.

As a first step, all modes are compiled, which is necessary for the Modelica simulation environments.

When all modes are compiled, the simulation parameters (determining start time, stop time, solver, etc.) are set in the framework.

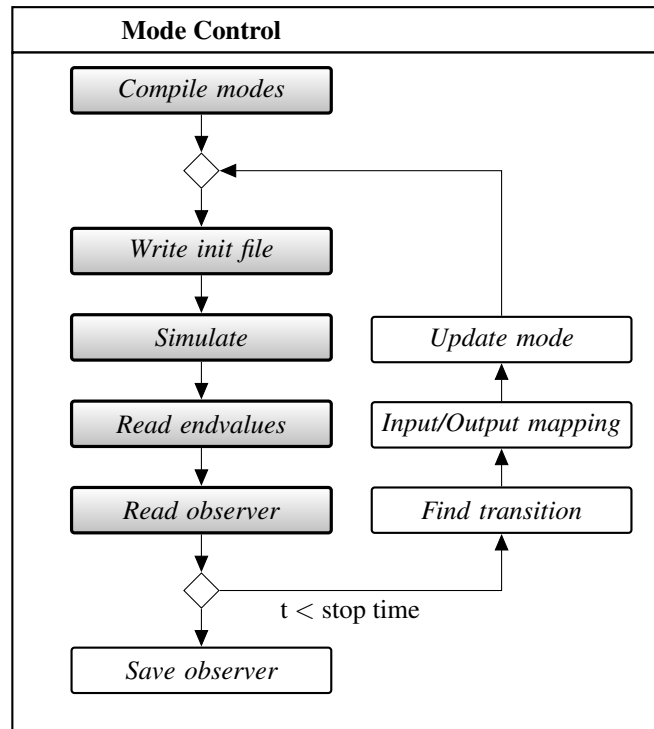


Figure 1: Schematic view of the steps to simulate a variable-structure model in DySMo

After this initial setup, the simulation of the first mode, using the appropriate executable and initialization file, is started. Each mode contains a certain number of stop conditions, which define when a simulation is terminated. When such a stop condition is reached, a variable *switch\_to* is set, which indicates the next mode of the simulation. Then the simulation terminates. The framework now reads the simulation results of the recently terminated simulation, and maps those endvalues to the starting values of the variables of the subsequent mode. Finally, the mode is updated to the new mode and a new simulation is started.

When the final stop time of the simulation is reached, the simulation terminates and the saved sim-

ulation data is stored in a file.

Using DySMo allows us to introduce variable structure to an existing model without having to completely re-model it, as only minor changes have to be made to add the stop conditions and make the desired adjustments to the equation sets of each mode. Furthermore, in certain modes we are able to greatly reduce the state space to only hold the needed information. This approach shown here is of course only suitable and sensible for models with a small number of modes. If many mode switches occur this framework is not suitable and in the future, creating and simulating variable-structure models should be integrated into Modelica or other languages. But since this is not the case yet, we want to illustrate the positive effects these kinds of models have with the DySMo prototype.

### 3 Rocket Model

In this section we introduce the classical as well as the redesigned model in greater detail. Both models were implemented in Modelica and simulated using Dymola. The object-oriented, component-based nature of Modelica allowed us to separate our model into different components which facilitated the variable-structure redesign.

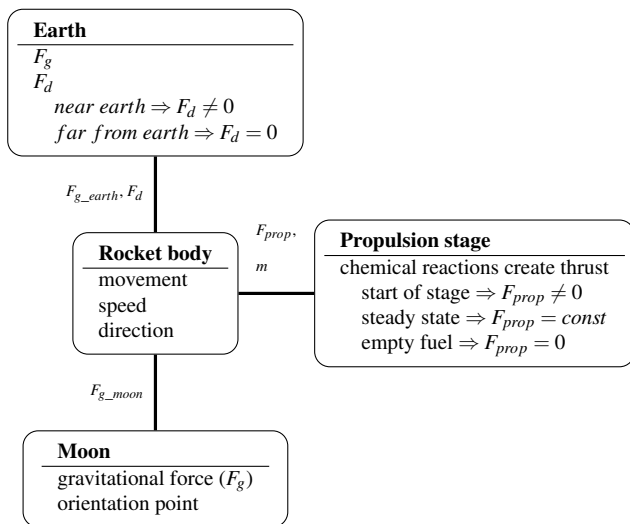


Figure 2: Components of the rocket model and properties that are calculated by them

**Basic mathematical description of the rocket**  
 Since the complete Model consists of many equations, it would be too long to explain all equations in detail. However we give a short overview about the most

important physical laws necessary to create the rocket model.

As the main interest was the trajectory of the rocket, we needed Newton's law of motion

$$F(s, t) = F_{prop} + F_g - F_d = m \cdot \ddot{s}.$$

The remaining task was to determine  $F$ . In our model the force consists of three parts. The aerodynamic drag ( $F_d$ ), the gravitational force ( $F_g$ ) and the propulsive power ( $F_{prop}$ ). The gravitation force is dependent on the planets (*Earth* and *Moon*) and their masses. Also the position and mass of the rocket have an influence.

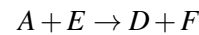
The aerodynamic drag is essentially

$$F_d = \frac{1}{2} c_w \rho A v^2$$

and is only calculated while the rocket is still in the atmosphere of the earth.

The remaining force is calculated by simulating the combustion of hydrogen in the combustion chamber. Therefore many thermodynamical and chemical laws come into play. We will mention only a few.

It is possible to model the change of concentration during a chemical reaction



with ordinary differential equations

$$\begin{aligned} \dot{C}_A &= -k C_A C_E, \\ \dot{C}_D &= k C_A C_E \text{ etc.} \end{aligned}$$

where

$$k = B T^{n_f} \exp\left(-\frac{E}{RT}\right).$$

Moreover we assume the ideal gas law

$$p = n \frac{RT}{V}.$$

It is then possible to determine the temperature using

$$\rho c_v \frac{dT}{dt} = \sum_{k=1}^r (-\Delta u_k) w_k$$

plus vaporization and heating terms. Here  $\Delta u_k$  is the reaction energy of reaction  $k$ . For a chemical reaction  $i$  where two substances  $A$  and  $B$  react, the coefficient  $w_i$  is defined as  $w_i := k_i C_A C_B$ . This energy leads to pressure which leads to a massflow through a throttle with

a certain velocity. At the end the propulsive power can be calculated with

$$F_{prop} = \dot{m}v.$$

The original and the variable-structure model were separated into different components, namely the rocket body, the propulsion stage and two celestial bodies, as shown in Figure 2. The figure already gives a short overview of what different calculations are necessary in each component.

### 3.1 Original Model

Basically, the rocket body calculates the movement of the rocket by taking into account the forces it is exposed to. The propulsion stage is responsible to calculate the thrust the rocket produces. It contains three booster modules that are used in succession. Each of the booster modules has a specified amount of fuel available and calculates the chemical reactions for the combustion of the solid propellant. This combustion builds up a pressure in the combustion chamber, through which the thrust of the rocket is determined. After starting a new stage the thrust is build up quite fast and afterwards is almost constant. These chemical reactions necessitate the solver to employ very small step sizes, as they take place very rapidly. Whenever the fuel of a booster module runs out, the propulsion stage will reinitialize using the next booster module available. Once the last booster module is emptied, the rocket will no longer generate thrust.

The celestial bodies are used to represent the moon and the earth. They supply the gravitational forces influencing the rocket. Air resistance and wind are also calculated in the earth component, as the rocket is subjected to these while passing through the atmosphere of earth. The moon, however, has no atmosphere, but is used as orientation point for the rocket.

All calculated forces are passed to the rocket body, where the movement of the rocket is calculated. The mass of the rocket changes substantially during the course of the simulation, as fuel is burned up and booster modules are discarded. The current mass of the rocket is determined by adding up the masses of the rocket body component, all remaining booster modules and the remaining fuel. When compared to the chemical reactions calculated in the propulsion stage of the model, the position of the rocket changes at a very slow pace.

The original rocket model takes about 50 seconds to simulate with the Dassel solver. The solver has to resort to a very small step size in order to simulate the

propulsion stage accurately, even though the propulsion is no longer needed after a steady state is reached or once the rocket runs out of fuel. As each simulation step generates data for all variables of the model, the overall data volume of a simulation measures about 1.5GB and postprocessing this data becomes rather tedious work.

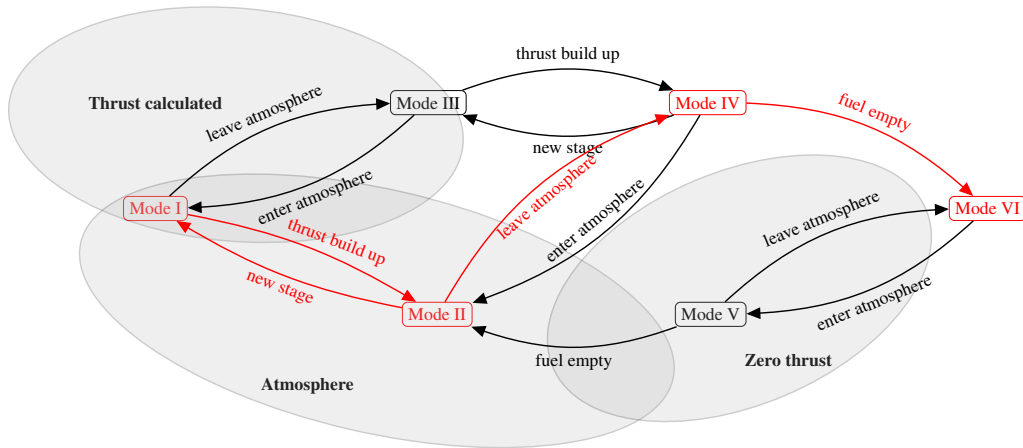
### 3.2 Variable-structure Redesign

When introducing variable structure to a model, the first step is always to identify the modes that are to be implemented, or, to put it another way, to determine in which way the model could benefit from variable structure. As already discussed, the rocket model has a stiff equation set, which necessitates very small step sizes for the simulation to be reasonably accurate. However, only during the ignition of each booster module it is necessary to calculate the chemical reactions inside the combustion chamber with such accuracy. Afterwards, the thrust is almost constant and regarding the chemical reactions is not necessary anymore.

Furthermore, the rocket has to travel a great distance without any thrust at all to reach the moon. The result is that the simulation runs with a very small step size for a long time while the variables feature only minimal changes.

The aim of our redesign is to eliminate this problem by holding the thrust constant once it has fully build up and take the equations for the chemical equations out of the model. After fuel runs out, the rocket either switches to another booster and builds up the thrust anew (regarding the chemical reactions), or it switches to a mode where thrust is assumed as zero if no further rocket stage is available. This leads to three different modes for the propulsion stage: building the thrust by regarding the chemical reactions, having constant thrust, or no thrust at all.

Another improvement is to calculate air resistance only while traveling through earth's atmosphere. This leads to two earth models, one that contains an atmosphere and one that does not. This sums up to six combinations of submodels, and therefore six modes, which are shown in Figure 3. Conditions for transitioning between the modes are also shown. At the bottom of the figure the actually performed mode switches from our model are listed. We can see that at the beginning of the simulation the rocket alternates between the modes I and II. This is due to the fact that only during the last propulsion stage of the rocket it is able to clear the atmosphere and therefore switch to



Mode switches: I -> II -> I -> II -> I -> II -> IV -> VI

Figure 3: Division of the rocket model into six submodels used as modes for the variable-structure redesign

mode IV, which has constant thrust and does not calculate the atmosphere anymore. This mode is active until the rocket's fuel is burnt up, in which case mode VI becomes active and sets the rocket thrust to zero for the remainder of the simulation.

It is noteworthy that, considering the given model, the modes III and V are never used. For mode III to be active the rocket would have to leave the atmosphere when it is in mode I. Meaning that the thrust of the current stage is not build up yet and the rocket left the atmosphere. With the given set of parameters this does not occur and therefore mode III is never used. Mode V is not used either, it would require the rocket to run out of fuel before it leaves the atmosphere, which it does not. We modeled these modes nonetheless, as they could be relevant if the simulation parameters were altered (consider for example starting from the moon which would necessitate mode III).

For the chosen approach, all six modes had to be implemented in Modelica, since Modelica does not allow to specify the necessary changes inside the submodels. Switching conditions were added to each mode, which define the modeID of the next mode and terminate the current simulation. In the DySMo framework the mode switches then have to be defined. This means that the initialization for each transition has to be regarded. For convenience, the modes for the rocket were implemented in such a way that variables are called the same if possible and therefore only a name matching is necessary for each switch. The framework then handles the occurring mode switches automatically and saves the simulation data. Each mode is only compiled once and then the executable is used in case

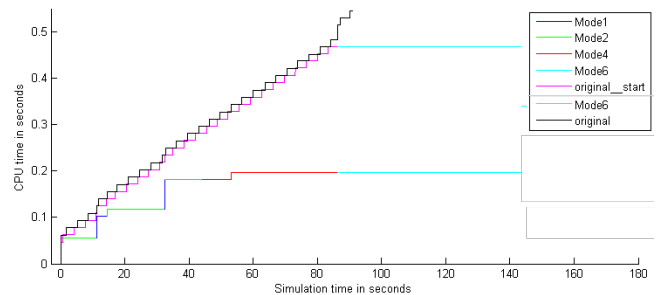


Figure 4: CPU time needed for the simulation in Dymola

the mode needs to be simulated again (as modes I and II are). This saves execution time since less compilations are necessary and therefore less communication with Dymola.

## 4 Results and Discussion

By redesigning the rocket model in a way that takes advantage of variable-structure modeling we intended to reduce the stiffness of the equation set, to speed up the simulation and to reduce the excessive amount of generated data without losing accuracy. In this section, we will discuss if these goals were reached, as well as the drawbacks of our approach.

Figure 4 shows the needed CPU time of the simulation runs in Dymola. Only the first 180 seconds after the launch of the rocket are shown to visualize the CPU time of the mode switches.

Three different simulation runs can be seen in this figure.

1. original model (black)
2. model with modes I, II, IV and VI
3. model which starts with the original model until the fuel runs out

At the beginning the CPU time is basically the same, the slight alterations could not be avoided. As can be seen the model with the many mode switches is considerable faster than the other ones. This becomes evident when the first switch occurs and the thrust is considered constant.

As can be seen when mode VI is reached only about half the time was needed for the simulation. When mode VI is reached the CPU time's increase is rather insignificant. The CPU time needed is about 0.4 seconds for the entire simulation. The CPU time for the original model increases constantly and needs about 50 seconds for a simulation time of 150 000 seconds.

So far, we only took into account the needed time for the simulation, but we also have to account for the time it takes to switch modes and to compile the additional models. The overall time it takes to simulate the redesigned model including all necessary steps is 4 seconds, of which only 0.4 seconds consist of actually simulating the model. The majority of the time, 3.1 seconds, is used to compile the models in Dymola. The framework, employed to switch between modes, accounts for the remaining 0.5 seconds. Still, the necessary time to simulate was significantly reduced compared to the 50 seconds it takes to simulate the original model.

Since the longest time was necessary for the compilation and the major speed advantages were seen when mode VI is active, we build another model, which is basically the original model until the fuel runs out. Then a switch to the last mode occurs. The result is also shown in Figure 4. Here we can see that we lose time compared to the first variable-structure model, but when the fuel is empty the CPU time becomes almost constant. Since only two modes need to be compiled and only one mode switch occurs the overall time necessary for this model is about 2.5 seconds, with switching, compiling and CPU time. Which is even faster than the originally designed variable-structure model. This shows that choosing the correct modes is not an easy task and needs to be done carefully.

Even though the needed time to simulate the rocket launch could be reduced, if the simulation results would be less accurate this improvement would lose meaning. In Figure 5 we can see two of the simulation

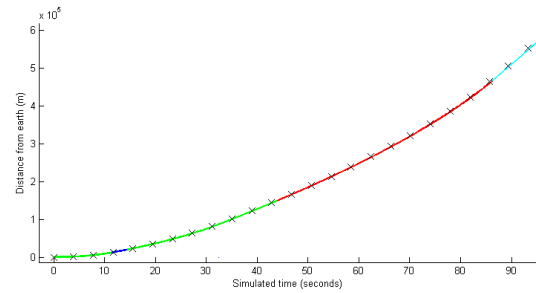


Figure 5: Rocket position comparison

results plotted on top of one another (the third simulation did show the same behavior). It is evident that the difference between both results is not discernible. Both results are essentially the same.

Another issue we had with the original model was the huge volume of generated data, 1.5GB, which resulted from storing variable data of every time step combined with the very small step sizes necessary to maintain accuracy for the stiff equation system. When simulating the redesigned model we are left with 8 result files; one for the simulation of each mode. When combined, the size of those files is about 1MB of information. Using the two mode model the simulation data of the two result files adds up to 30MB. It is evident that the major part of the original data was not needed to accurately calculate the trajectory of the rocket.

Dymola does support an option to choose the amount of data to be stored during a simulation run, but changing the setting for the original model resulted in failed simulations. It seems that this setting has an effect on the solver in Dymola. Imagine it would be possible to choose the amount of stored data: Due to the different time scales of the model we would have to find a compromise between storing many values of very fast changing variables while the thrust is built up and less values of slow changing variables during the flight to the moon with maximum thrust. With this compromise we could reduce the amount of data but may also lose information about data of the chemical reactions.

A problem we encountered with the variable-structure model was that the solver of Dymola seems to be influenced by the start time of the model. Whenever we started a simulation at a time other than zero, the simulation failed. The rocket model does not depend on the starting time, as the *time* variable is not used in the model. Thus, the solver should have been able to start at any given time and still calculate the same results. This was not the case for this model though. Therefore, we had to start each mode's simu-



lation at zero seconds and later add a time offset to get correct results.

## 5 Summary

In this paper we have shown that modeling with variable structure is worthwhile. We accomplished our goal to lessen the necessary time to simulate the flight to the moon without loosing accuracy and were able to significantly reduce the volume of generated data. We have also shown that it is not a trivial task to choose good modes and that when choosing different modes it is even possible to save more simulation time without disadvantages. Using variable-structure models seem to be a good choice for stiff equations in case the stiffness can be taken out for parts of the simulation.

## References

- [1] Felix Breitenecker. Development of simulation software - from simple ode modelling to structural dynamic systems. In L. Louca, Y. Chrysanthou, Z. Oplatková, and K. Al-Begain, editors, *Proceedings of the 22nd European Conference on Modelling and Simulation (ECMS 2008)*, pages 20–37, 2008.
- [2] Hilding Elmqvist, François E. Cellier, and Martin Otter. Object-oriented modeling of hybrid systems. In *Proceedings of ESS'93, European Simulation Symposium*, pages 31–41, 1993.
- [3] Alexandra Mehlhase. A python framework to create and simulate models with variable structure in common simulation environments. *To be published in Mathematical and Computer Modelling of Dynamical Systems*, 2013.
- [4] Henrik Nilsson and George Giorgidze. Exploiting structural dynamism in functional hybrid modelling for simulation of ideal diodes. In *Proceedings of the 7th EUROSIM Congress on Modelling and Simulation, Prague, Czech Republic*. Czech Technical University Publishing House, 2010.
- [5] Christoph Nytsch-Geusen, Thilo Ernst, André Nordwig, and et al. Mosilab: Development of a modelica based generic simulation tool supporting model structural dynamics. In Gerhard Schmitz, editor, *Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005*, pages 527–535. TU Hamburg-Harburg, 2005.
- [6] Tuncer Ören. Model update: A model specification formalism with a generalized view of discontinuity. In *In Proceedings of the Summer Computer Simulation Conference*, pages 689–694. Springer-Verlag, 1987.
- [7] Thorsten Pawletta, Bernhard Lampe, Sven Pawletta, and Wolfgang Drewelow. A devs-based approach for modeling and simulation of hybrid variable structure systems. In Sebastian Engell, Goran Frehse, and Eckehard Schnieder, editors, *Modelling, Analysis, and Design of Hybrid Systems*, volume 279 of *Lecture Notes in Control and Information Sciences*, pages 107–129. Springer Berlin Heidelberg, 2002.
- [8] Levent Yilmaz and Tuncer I Ören. Dynamic model updating in simulation with multimodels: A taxonomy and a generic agent-based architecture. In *Proceedings of SCSC 2004 - Summer Computer Simulation Conference*, pages 25–29, 1998.
- [9] Bernard P. Zeigler. *Theory of Modeling and Simulation*. John Wiley, 1976.
- [10] Dirk Zimmer. *Equation-based modeling of variable-structure systems*. PhD thesis, Eidgenössische Technische Hochschule ETH Zürich, Switzerland, 2010.



# Efficient Monte Carlo simulation of stochastic hybrid systems

Marc Bouissou<sup>1,5</sup>, Hilding Elmqvist<sup>2</sup>, Martin Otter<sup>3</sup>, Albert Benveniste<sup>4</sup>

<sup>1</sup>EDF R&D, 1 av. du Général de Gaulle, 92141 Clamart, France

<sup>2</sup>Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden

<sup>3</sup>DLR, Institute of System Dynamics and Control, D-82234 Wessling, Germany,

<sup>4</sup>IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cédex, France

<sup>5</sup>Ecole Centrale Paris, Grande Voie des Vignes, 92295 Châtenay Malabry, France

[Marc.Bouissou@edf.fr](mailto:Marc.Bouissou@edf.fr), [Hilding.Elmqvist@3ds.com](mailto:Hilding.Elmqvist@3ds.com), [Martin.Otter@dlr.de](mailto:Martin.Otter@dlr.de), [Albert.Benveniste@inria.fr](mailto:Albert.Benveniste@inria.fr)

## Abstract

This paper proposes an efficient approach to model stochastic hybrid systems and to implement Monte Carlo simulation for such models, thus allowing the calculation of various probabilistic indicators: reliability, availability, average production, life cycle cost etc. Stochastic hybrid systems can be considered, most of the time, as Piecewise Deterministic Markov Processes (PDMP). Although PDMP have been long ago formalized and studied from a theoretical point of view by Davis (*Davis 1993*), they are still difficult to use in real applications. The solution proposed here relies on a novel method to handle the case when the hazard rate of a transition  $\lambda$  depends on continuous variables of the system model, the use of an extension of Modelica 3.3 and on Monte Carlo simulation. We illustrate the approach with a simple example: a heating system subject to failures, for which we give the details of the modeling and some calculation results. We compare our ideas to other approaches reported in the literature.

*Keywords: Stochastic hybrid system; PDMP; dynamic reliability; state-dependent hazard rate; continuous time state-machine; Monte Carlo Simulation.*

## 1 Introduction

Usually, Modelica models are deterministic; they are built to simulate the nominal behavior of the systems they represent. In order to challenge the functioning of these systems in diverse situations, or in the presence of a varying environment, a degree of randomness is sometimes added to the system inputs.

But the kind of models this paper is dedicated to is quite different: here, *the random behavior can be due to the system itself, mainly because of failures and repairs of components*. The purpose of reliability, and more generally, of dependability studies is to

calculate probabilities of undesirable events such as the failure of the mission of a system, or to estimate the probability distribution of some performances of the system: total production on a given time interval, maintenance cost, number of repairs etc. Usually, dependability studies are performed with dedicated methods and tools, based on discrete (and often even Boolean) models of systems: fault trees, Markov chains, Petri nets, etc.

However, in some situations, a purely discrete representation of a system cannot be a good enough approximation: this is the case of hybrid systems, having both discrete and continuous parts, with strong interactions between the two. Reliability analysts call the study of such systems "dynamic reliability".

Below are two examples showing the need for powerful tools for dynamic reliability studies:

In the probabilistic safety analysis of nuclear power plants, so-called "level 1" studies, that are those aiming at assessing the probability of a core melt, rely on discrete (mainly Boolean) models. But after a core melt, components are subject to physical stresses (temperature, humidity and radioactivity) that may modify their behavior and increase very much their failure rates, and this should be taken into account in "level 2" studies that try to assess the risks of radioactivity release in the environment. Even for level 1 studies, there is a competition in time between the decay of the thermal power that must be evacuated and the failures of components, and the coarse decomposition of scenarios according to large time intervals in which failures can happen can be excessively pessimistic.

Another example of system associated to very high stakes and whose behavior cannot be captured correctly without a stochastic *hybrid* model is the electrical grid. In this system, transients with extremely different time scales can happen and evolve to blackout situations. For example, after the failure of a line, the intensity increases in the remaining

lines; it can augment their temperature which, because of dilatation, makes them come closer to the ground. This increases the probability of a new fault due to a contact between a line and a tree.

Recent results have shown that numerical schemes can solve PDMP with a small size, however Monte Carlo Simulation remains the only possible method for quantification in larger cases.

The purpose of this paper is twofold: it shows how to *model* a hybrid stochastic system and it gives an *efficient* Monte Carlo scheme that *works even in the case of failure rates varying with the continuous variables of the system*.

## 2 A review of formalisms used for hybrid system modeling

In this section, we will first give the theoretical model called PDMP (Piecewise Deterministic Markov Processes). Then we will explain the limits of PDMP and why these limits are not a real problem as long as we intend to model *physical* systems (and not, for example, financial systems). Finally we will see how some existing formalisms (including Modelica) can be used to represent PDMP.

### 2.1 The theoretical model PDMP

The state at time  $t$   $(x(t), m(t))_{t \geq 0}$  of a hybrid system is composed of two parts: a continuous one,  $x(t) \in \mathbb{R}^n$  and a discrete one,  $m(t) \in \mathbb{N}$ .

$x(t)$  usually models physical variables such as temperature, pressure, volume, flow rate, etc., whereas  $m(t)$  is the index of the state of the discrete "part" of the system: to each value of  $m(t)$ , one can associate discrete states (such as working or failed, open or closed etc.) to each component of the system.

What makes the resolution of dynamic reliability problems difficult is the existence of bi-directional interactions between  $x(t)$  and  $m(t)$ . Here are some examples of such interactions:

- $x(t)$  acts on  $m(t)$ . When a physical variable reaches a threshold, it can provoke a discrete change: explosion of a tank because of high pressure, evaporation of steam because of high temperature, reaction of the instrumentation and control system. A physical variable can also make a discrete event happen earlier or later: for example, a failure rate may increase with the temperature.
- $m(t)$  acts on  $x(t)$ . The opening or closure of a valve, the failure of a pump changes the differential equations governing physical variables.

From a mathematical point of view, PDMP contain all the ingredients needed to model stochastic hybrid systems such as those exemplified above (Davis 1993).

The general equations governing the evolution of the PDMP whose state is described by  $(x(t), m(t))_{t \geq 0}$

$$\frac{dx(t)}{dt} = g(x(t), m(t))$$

$$\Pr(m(t + \Delta t) = j / m(t) = i, x(t)) = \lambda(i, j, x(t)) + o(\Delta t)$$

Here,  $\lambda(i, j, x(t))$  denotes a function that defines the hazard<sup>1</sup> rate from state  $i$  to state  $j$  for the discrete part of the system:  $\mathbb{N} \times \mathbb{N} \times \mathbb{R}^n \xrightarrow{\lambda} \mathbb{R}$ . In other words, it defines the probability that a transition occurs from discrete state  $i$  to state  $j$ .

### 2.2 Scope and limits of PDMP

The scope of PDMP is quite large: it generalizes all discrete models used in dependability analysis, even those considered as *non markovian*, (like for example Petri nets with arbitrary probability distributions for delayed transitions) due to the modeling "trick" explained below.

Thanks to the insertion into  $x(t)$  of the time elapsed since the beginning of the life of an aging component, it is possible to model the probability distribution of the time to failure of this component, whatever this distribution may be.

For example, here is how we can transform a non markovian process with two states modeling a component with a Weibull distributed lifetime into a markovian process, thanks to the addition of time in the definition of the state:

- the "usual" definition ( $m = 1$  corresponds to a working state, and  $m = 0$  corresponds to a failed state):

$$\Pr(m(t) = 0) = 1 - \exp\left(-\left(\frac{t}{\alpha}\right)^\beta\right) \quad (1)$$

In this expression,  $\alpha \in \mathbb{R}^{+*}$   $\alpha$  is the scale factor and  $\beta \in \mathbb{R}^{+*}$  is the shape parameter of the Weibull distribution.

- definition with a PDMP whose continuous variable represents the time:

$$\frac{dx(t)}{dt} = 1 \quad \text{because } x = t \quad (2)$$

<sup>1</sup> Here, reliability analysts would rather use the term "transition rate" instead of "hazard rate". These two expressions are synonyms, but we use the second one because it is the most neutral. It is the quantity defined in eq. (6).

$$\Pr(m(t + \Delta t) = 0 / m(t) = 1, t) = \frac{\beta}{\alpha} \left( \frac{t}{\alpha} \right)^{\beta-1} \exp\left(-\left(\frac{t}{\alpha}\right)^\beta\right) + o(\Delta t) \quad (3)$$

$$\Pr(m(t + \Delta t) = 0 / m(t) = 0, t) = 1 \quad (4)$$

In equation (3), we use the *hazard rate* of the Weibull distribution.

This kind of representation by a PDMP can be generalized to any lifetime distribution; the remarkable case when the hazard rate is in fact constant corresponds to the exponential distribution (see section 3.1).

The large expressive power of PDMP unfortunately comes with a heavy additional burden for analysts: as one can see from the very elementary example given above, PDMP are not at all easy to manipulate. In fact, they are both difficult to specify, and to solve by methods other than Monte Carlo simulation.

How about their limits? Of course, PDMP do not address all the needs for reliability studies of systems involving uncertain dynamics. Neither random continuous inputs nor measurement noise can be captured. Still, PDMP offer a first interesting step beyond classical dynamic dependability models with discrete space. PDMP are interesting in that they do not require the modeling and simulation of full fledge stochastic differential equations. Their Monte-Carlo simulation can be performed at reduced cost, as we shall see.

### 2.3 Modeling in practice

Modeling hybrid systems has long been a concern for the study of purely deterministic systems.

For relatively simple models, the graphical representations used in control and signal processing can suffice. They allow the graphical construction of transfer functions, using assemblies of elementary blocks representing integrators, differentiators, multipliers, adders, thresholds etc.

For more complex models, a higher level of abstraction is needed. This can be achieved by the encapsulation of algebraic, differential and discrete equations in objects corresponding to physical components. This is the solution made possible by Modelica. Thanks to Modelica libraries, it is possible to quickly build models of mechanical, electrical, fluid etc. systems, encompassing thousands of equations. However, so far this kind of representation has not been extended to allow a convenient modeling of **stochastic** hybrid systems.

Thanks to a comparison between various existing modeling languages for PDMP done in (Bouissou

and Jankovic 2012) and (Bouissou et. al. 2013), the missing features in Modelica 3.3 can be identified:

- Asynchronous state machines in which transitions are triggered by events (instead of synchronous state machines triggered by a clock)
- Transitions that can be associated to random delays (this is the most important and delicate point)
- Allowing several instantaneous transitions from one state having specified probabilities of firing.

Section 4 will describe the two first points in detail<sup>2</sup>, but before that, we will give what is in fact the main point of this paper: a smart algorithm allowing to perform Monte Carlo simulation on PDMP. This algorithm is usable whatever the interactions between the discrete and continuous parts of the process and is very economical in terms of CPU usage.

## 3 Making the Monte Carlo simulation of a PDMP efficient

### 3.1 State of the art

The state of the art Monte Carlo simulation method for PDMP is described in (Zhang et al. 2008). This paper recalls the mathematical definition of PDMP as it was set up by Davis and gives an iterative simulation algorithm that determines the successive times of process jumps due either to a random event or to the fact that the continuous part of the system reaches the boundary of the currently valid domain for the differential equations.

Starting from the initial state of the system, the first jump date is the minimum of the dates of the set of events corresponding to:

- Boundaries crossings: the corresponding dates are obtained by solving the current set of differential equations until one of the continuous variables crosses a threshold;
- Random events. In most cases, there is a competition between several transitions associated to individual probability distributions of the times to firing of these transitions (like in a stochastic Petri net). For example, several components can fail at any moment, but if their failures are independent, one will fail first, and this will determine the instant of the first jump date.

When the random processes are independent from continuous variables, it is easy to determine, at  $t=0$ ,

<sup>2</sup> The third point will *not* be developed in this article, both because of a lack of space and because it is not related to the two other points.

the dates of all random events (details given hereafter). But if they are not, their determination is more difficult. We will first recall the definition of the hazard rate, associated to the distribution of any random variable such as the time to a failure or a repair then explain a method able to find in one run, without any backtrack, the date of the first event in the system, whatever its nature (random or boundary crossing).

Given a random time  $T$  whose *cumulative distribution function (cdf)*  $F$  is defined as

$$F(t) = \Pr(T < t) \quad (5)$$

the corresponding *hazard rate*,  $\lambda(t)$ , is defined as:

$$\lambda(t) = \lim_{\Delta t \rightarrow 0} \frac{\Pr(T < t + \Delta t | T > t)}{\Delta t} \quad (6)$$

The hazard rate can then be expressed as

$$\lambda(t) = \frac{F'(t)}{1 - F(t)} \quad (7)$$

that is

$$\frac{dF(t)}{dt} = (1 - F(t))\lambda(t) \quad (8)$$

For Monte Carlo simulation, the time to the next event,  $T$ , is determined by drawing a uniform random number,  $r$  in  $[0,1]$ , and solving:

$$F(T) = r$$

When  $\lambda$  is constant, the solution to the differential equation (7) is:

$$F(t) = 1 - e^{-\lambda t}$$

and

$$T = -\frac{\ln(1 - r)}{\lambda} \quad (9)$$

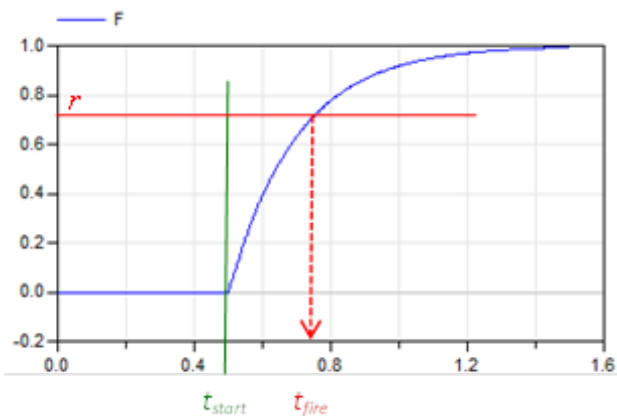


Figure 1: The "inverse cdf" technique for drawing a random number according to a given distribution

In figure 1, this approach is visualized (here:  $t_{start}$  is the time instant when the random number  $r$  is drawn,

and  $t_{fire}$  is the time instant of the stochastic event, so  $T = t_{fire} - t_{start}$ , the blue curve is the cumulative distribution function  $F$ ).

When the hazard rate of a transition depends on continuous variables  $x$ , so,  $\lambda(t, x(t))$ , the cumulative distribution function  $F$  can be obtained by integration of eq. (7) and is equal to

$$F(t) = 1 - e^{-\int_0^t \lambda(u) du} \quad (9)$$

Then, the usual and "naïve" way to proceed is to integrate the differential equations up to a "large enough" time, draw a uniform random number  $r$  and calculate  $T$  as  $F^{-1}(r)$ .

This process involves solving the differential equations of the model during a sufficiently large time interval and calculation of  $\lambda(t)$  which is dependent on variables of the model. A numerical method is then used for integration of  $\lambda(t)$  followed by calculation of  $F(t)$ . After this, the integrator of the differential equations needs to be rewound to  $t_{fire}$ , an operation normally not present in numerical integration methods.

### 3.2 New method for state dependent hazard rates

Instead of this complex and slow procedure, a new method has been developed that utilizes the fact that  $F$  is monotonically increasing and the zero crossing solver for events available in modern integration routines can be used to find the next event time  $t_{e,i+1}$ . The complete set of equations can be defined in the following way:

**At event**  $t = t_{e,i}$ :

$$\begin{aligned} m(t_{e,i}) &= g(m(t_{e,i-1}), t_{e,i}) \\ F(t_{e,i}) &= 0 \\ r(t_{e,i}) &= \text{random}() \end{aligned}$$

**for**  $t_{e,i} < t < t_{e,i+1}$ :

$$\begin{aligned} 0 &= f(\dot{x}, x, y, t, m(t_{e,i})) \\ \dot{F} &= (1 - F) \cdot \lambda(\dot{x}, x, y, t, m(t_{e,i})) \\ t_{e,i+1} &= \min_{t > t_{e,i}} t, \text{ such that } F \geq r(t_{e,i}) \end{aligned}$$

**where**

$$t \in \mathbb{R}, x(t) \in \mathbb{R}^{n_x}, y(t) \in \mathbb{R}^{n_y}, F(t) \in \mathbb{R}$$

$$m(t_{e,i}) \in \mathbb{N}, f(\dots) \in \mathbb{R}^{n_x + n_y}, \lambda(\dots) \in \mathbb{R}$$

This system consists of a continuous-time DAE (Differential Algebraic Equation system) defining the physical model, together with a state machine. The active state of the state machine is characterized by

the integer variable  $m$ . The DAE is a function of this active state  $m$ , and of continuous-time states  $x$  and algebraic variables  $y$ .

At an event instant  $t = t_{e,i}$  a transition to the next state of the state machine occurs, the cumulative distribution function  $F$  is re-initialized to zero, and a random number  $r$  is drawn.

Afterwards the DAE together with the differential equation for  $F$  is integrated until

$$F(t) - r(t_{e,i}) = 0$$

This means that the zero crossing of  $F(t) - r(t_{e,i})$  triggers a state event<sup>3</sup> and the corresponding (stochastically determined) time instant is the next event instant  $t_{e,i+1}$ .

For notational convenience this description was given for a special case. It is easy to generalize for several state machines where one or more stochastic and/or deterministic transitions are defined at the active states.

Remark: the above approach can be seen as a generalization of the simulation of nonhomogeneous Poisson processes (Sheldon 1990). Indeed, two methods of generating a Poisson process with known time-varying hazard rate function  $\lambda(t)$  are proposed in Section 5.5 of this reference, of which the second one can be seen as a basis of our method: it is proposed in Section 5.5 to compute

$$F(t - t_e) = 1 - e^{-\int_0^{t-t_e} \lambda(u) du}$$

and then to invert the equation  $F(t - t_e) = r(t_e)$  for  $t$ , where  $r$  is a random number drawn at the last event time  $t_e$ . We propose to differentiate the above equation, thus making clear that the time-varying intensity  $\lambda(t)$  can be given on-line. With this observation, we can now allow that  $\lambda(\dot{x}, x, y, t, m)$  is a function of the variables of a DAE describing the physical system,  $F$  is computed by integrating the differential equation for  $F$  together with the system DAE, and the stochastic event time is computed as the state event where  $F \geq r$  becomes true. We discovered our method independently, however, and this reference was subsequently pointed to us by Pierre Brémaud.

## 4 Modeling PDMP in Modelica

In this section it is shown how PDMP can be modeled in Modelica and how Monte Carlo simulations can be carried out over such models. Furthermore,

<sup>3</sup> State events are supported by modern ODE and DAE solvers; the solver will automatically iterate around the time instant where this function crosses zero, will backtrack, and will find the event time up to a certain precision.

with the novel technique from section 3.2 it is possible to model hazard rates that depend on the states of continuous variables in an efficient way.

### 4.1 Overview

In Modelica 3.3 support for hierarchical, synchronous, clocked, state machines (Elmqvist et al. 2012) was added. Such state machines are evaluated at clock ticks of sampled data systems, and are now the preferred way to model state machines in Modelica. In a prototype of Dymola, this state machine type was extended to model also continuous-time state machines (Elmqvist et al. 2014) and is the basis for the PDMP implementation in Modelica.

The basic mechanism is a generalization of transitions in the Modelica synchronous state machines, as sketched in the following figure:

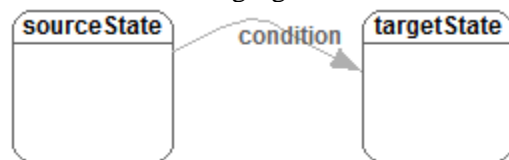


Figure 2: Transition between two states in a state machine

A transition in a Modelica 3.3 synchronous state machine fires, if its source state is “active” and the transition “condition” becomes true. If transition flag “immediate=true”, the transition fires immediately (so at the same clock tick). If “immediate=false”, the actual firing occurs at the next clock tick, so it is delayed. This approach is generalized in the following way for continuous-time asynchronous state machines:

- If “immediate=true”, the behavior is as before, so the transition fires immediately (at the current model evaluation).
- If “immediate=false”, by default a new event iteration is triggered and the transition fires at the next event iteration (so with an infinitesimal small delay that breaks algebraic loops).

There are several useful ways to delay a transition, such as by a fixed time period (e.g. firing after 2ms), or by a time period that is defined stochastically in different ways (as needed for a PDMP). Since there are many possibilities, it seems not possible to pre-define this behavior in a language element, but it needs to be configurable by a user.

In the following sub-sections, the different ingredients needed for such a Modelica extension are discussed.

### 4.2 Random number generation

In order to draw random numbers for the triggering of stochastic transitions, a random number generator is needed. A standard random number generator in a programming language is an impure function and is typically called as “`r = random()`”, so the function has no arguments and returns for every call a different random number, typically in the range  $0 \leq r \leq 1$ . It is clear that such a function cannot be implemented as a Modelica function, because Modelica functions are “pure”, and return always the same value, when called with the same input arguments. There are the following remedies:

- (1) Explicitly pass the internal memory (usually called “seed”) of the random number function as input and output arguments:

```
(r, seed) = random(pre(seed))
```

As a result, the random function can return a different (`r, seed`) value only at an event instant, as it should be, because the operator `pre(..)` can only be used in a discrete equation. It is therefore guaranteed that the random number function cannot be called during continuous integration which would give severe problems with the integration method.

- (2) Use an external Modelica function as interface to a C-function, `r = random()`, and mark this function as impure:

```
impure function random
  output Real r;
  external "C" r = random();
end random;
```

The “`impure`” keyword introduced in Modelica 3.3 guarantees that the random function can be basically only called in a when-clause, so at event instants.

Calling a random number generator in any simulation environment is tricky, because there are different requirements and depending on the analysis, simulation runs should (a) use different random number sequences in every simulation run, as for Monte Carlo simulations, or (b) should use the same random number sequences in specific simulations, as for Optimization over Monte Carlo simulations (Looye, Joos 2006) or when developing a model.

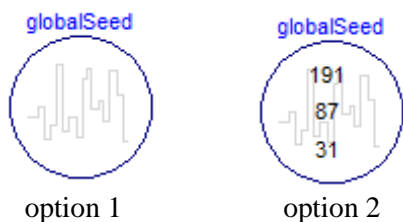


Figure 3: Two options for the pseudo random generator

Taking into account the above observations, a Modelica model was designed to define the random properties globally: The user has to drag model “GlobalSeed” in the model and can then select from two options: in the first case (by default), for every simulation run a different initial seed is selected. In the second case, defined by a flag, initial seeds can be explicitly defined and every simulation run will use the same random number sequence. The selected seeds are displayed in the icon.

In a model, the random number generation function is called in the following way:

```
protected
  outer GlobalSeed globalSeed;
  Real r "random number";
equation
  when condition then
    r = globalSeed.random();
  end when;
```

Since `globalSeed.random` is defined as an impure function, it can only be called in a when clause, so only at an event instant. Every call returns a different random number in the range  $0 \leq r \leq 1$ .

### 4.3 Delayed transition blocks

As already mentioned, there are many ways to define the delay of a transition. In order to keep this user configurable, it is proposed to define the delay by a Modelica block, with one Boolean input and one Boolean output signal, which has the following properties:

The input signal of such a block, called `enableFire`, signals when the source state is active and the transition condition becomes true. The signal is then set to `true`, until it is explicitly reset (either because the transition fired, or the source state became inactive, e.g., due to the earlier firing of another transition).

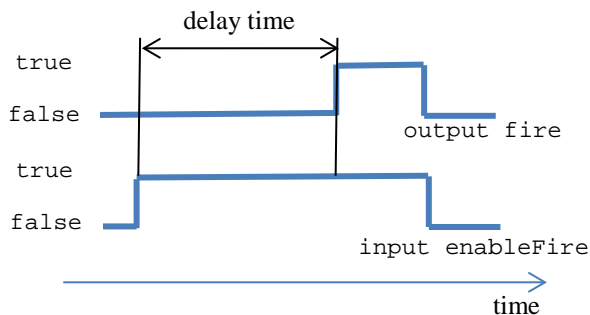


Figure 4: Behavior of a delay block

The delay block triggers an event after a defined “delay time” and at this time instant the output signal `fire` is set to `true`. This raising signal triggers the firing of the transition. Immediately, when the source



state becomes inactive, `enableFire` and `fire` are both reset to `false`.

Based on this principle, a small library of delay blocks has been implemented. The two deterministic delay blocks `FixedTimeDelay` and `VariableTimeDelay` define the delay by a fixed or variable deterministic time delay, respectively. The core part of the `FixedTimeDelay` block implementation is (`t_start` is the time instant when the input `enableFire` is rising<sup>4</sup>):

```
if enableFire then
  fire = time >= t_start + delayTime;
else
  fire = false;
end if;
```

Therefore, a time event is triggered after the defined `delayTime` and the output `fire` changes to `true` at this time instant. Conceptually, the equations in a transition are defined as:

```
algorithm
  when initial() then
    enableFire := stateActive and
                  condition;
  elseif stateActive and condition then
    enableFire := true;
  elseif not stateActive then
    enableFire := false;
  end when;

equation
  if immediate then
    fire = enableFire;
  else
    delayBlock.enableFire = enableFire;
    fire = pre(delayBlock.fire);
  end if;
```

Note, the output `fire` of the delay block needs to have an infinitesimal small delay via the `pre(...)` operator in order to break algebraic loops.

#### 4.4 Randomly delayed transitions

The approach sketched in section 3 for fixed (not state-dependent) hazard rates leads the following implementation in Modelica, where the implementation of eq. (9) can be easily identified:

```
outer GlobalSeed globalSeed;
Real r, t_next;
parameter Real hazardRate;

equation
  when enableFire then
    r = globalSeed.random();
    t_next = time - log(1-r)/hazardRate;
  end when;
```

<sup>4</sup> At the time instant where `enableFire` becomes true, the actual value of the variable delay time is inquired and this value is used as delay time.

```
if enableFire then
  fire = time >= t_next;
else
  fire = false;
end if;
```

Since the condition `time >= t_next` is a purely time dependent condition, a Modelica tool will determine the time instant of the fire time in advance and will directly (and therefore efficiently) integrate to this time instant.

Other stochastic distributions can be implemented in a similar way, provided that the distribution of the time to the firing is invariant once the `enableFire` Boolean has become true.

#### 4.5 State dependent randomly delayed transitions

In this section the implementation of the innovative approach from section 3.2 is sketched: the corresponding delay block can be defined in Modelica as:

```
outer GlobalSeed globalSeed;
Real r;
input Real hazardRate(min=0);
equation
  when enableFire then
    r = globalSeed.random();
    reinit(F,0); // start at F=0
  end when;

der(F) = (1-F)*hazardRate;
if enableFire then
  fire = F >= r;
else
  fire = false;
end if
```

The expression `F >= r` defines a state event and will therefore trigger a search process to determine the time instant when this condition becomes true up to a certain numerical precision. At this time instant an event is triggered.

#### 4.6 Asynchronous state machines

As already mentioned, in a prototype of Dymola the Modelica 3.3 synchronous state machines have been extended to continuous-time (= asynchronous) state machines. For all details see (Elmqvist et.al. 2014). As a very short summary: the states of a state machine might be non-clocked Modelica models or blocks. The “active” state is always simulated together with the rest of the DAE system and all outgoing transitions from this state are monitored. If one of these transitions fires, an event is triggered, the active state is changed and simulation continues with the new active state. All de-activated state models or blocks are “frozen”, so all variables in these objects

keep their values until these states become again “active”.

In Modelica 3.3 a transition is defined with the following built-in operator (see Modelica 3.3 specification, section 17.1):

```
transition(from, to, condition, immediate, reset,
           synchronize, priority)
```

Here “from” and “to” are the instance names of the blocks used as source and target state of the transition, “condition” is the firing condition and “immediate” is the flag that defines whether the transition is immediately firing or is firing at the next clock tick (the remaining arguments are not important for the following discussion).

This “transition” operator holds in principal also for continuous-time state machines. However, the case for “immediate=false” needs to be differently defined: As sketched in the previous sections, a wide variety of useful delay definitions can be provided by different blocks. Therefore, one approach would be to use a replaceable block as additional argument in the transition operator. Example:

```
transition(“state1”, “state2”, true, false, true, true, 1,
           redeclare FixedHazardRateDelay
           delay(hazardRate=0.03));
```

This call would use block “FixedHazardRateDelay” for the block instance “delay” with the given modifier. Built-in operators in Modelica have the syntax of a function call. However, a block, being replaceable or not, cannot be passed to a function, and therefore, this construct seems to be unnatural to a built-in operator. It would be possible to pass a function object, such as:

```
transition(“state1”, “state2”, true, false, true, true, 1,
           delay = function FixedHazardRateDelay
                 (hazardRate=0.03));
```

However, with a function object, it would not be possible to define stochastic transitions that depend on continuous-time states (see section 4.5) because a differential equation needs to be solved in the object and state events need to be triggered.

It is therefore proposed to define the transition built-in operator with a syntax that is close to the function object:

```
transition(“state1”, “state2”, true, false, true, true, 1,
           delay = block FixedHazardRateDelay
                 (hazardRate=0.03));
```

Informally, the semantics is that the provided block (here: FixedHazardRateDelay) with its modifier is instantiated in the scope of the source state, so this block is only active and running, when its source

state is active and otherwise is “frozen”. If a delay block is not explicitly given, the default block will just implement the equation

```
fire = pre(enableFire);
```

so an infinitesimal small delay will be introduced.

Unfortunately, such a built-in operator is uncommon in Modelica (having a replaceable block as an argument to a built-in operator) and it is unclear whether this complicated definition should be introduced into the Modelica language specification.

For this reason, another approach is used in the prototype: The desired delay block is instantiated inside the source state. In the transition, the output of the delay block is utilized. The implementation in the source state is basically straightforward by instantiating from the desired delay block:

```
FixedHazardRateDelay delay(
    enableFire=enteringState(),
    hazardRate=0.3);
```

An issue is to define, when the random number shall be drawn (that is, when the input enableFire has a rising edge). In the prototype, enableFire is defined to have a rising edge, when the source state is entered. This is inquired, with the (not yet standardized) built-in operator “enteringState()”, that is available as a prototype in Dymola. As transition condition simply “<source-state>.delay.y” is used.

## 5 Case study

### 5.1 Preamble

The test case we are going to use to illustrate our ideas was first used to screen a number of potential tools and approaches adapted to dynamic reliability (Bouissou and Jankovic 2012), (Bouissou et al 2013) It has the advantage that it is easy to understand.

Although this test-case is very simple it has been quite difficult to solve it with tools that were initially designed only for modeling deterministic systems (and this includes Modelica tools). We intend to solve a more complex test-case in the future, such as the well-known “heated tank” problem used by T. Aldemir in (Aldemir 1991), that has been solved since then with many different approaches e.g. (Lair et al. 2010), (Zhang et al. 2008), (Broy et al. 2011), (Zhang et al. 2013).

### 5.2 The “heated room” test case

Consider a room containing a heater. A temperature sensor with a hysteresis switches the heater on when the ambient temperature falls below 15°C and switches it off when the temperature reaches 20°C.

The outside temperature is constant: 13°C. At time  $t = 0$ , the temperature of the room is 17°C, and the heater is on.

The flow of energy (power) traversing the walls is proportional to the difference of temperature between the inside and the outside of the room. When the heater is on, it injects a constant power in the

$$\frac{dT}{dt} = 0.1 \times (\text{Outside\_temperature} - T) + 5 \times (\text{heater\_is\_on})$$

room. Let us suppose that the isolation of the room and the heater power are such that the differential equation giving the evolution of the temperature is as follows (with  $t$  in hours, and  $T$  in °C):

In this expression, *heater\_is\_on* is an indicator function, with the value 1 if the heater delivers power and 0 otherwise.

If the heater was not subject to failures, the trajectory of temperature as a function of time would be a deterministic succession of portions of exponential functions, alternatively convex and concave, "oscillating" between 15 and 20 °C.

But in fact, the heater has a constant failure rate  $\lambda = 0.01/h$ , and a constant repair rate  $\mu = 0.1/h$ . How does this random behaviour affect the evolution of the temperature?

## 6 Modelica models for the case study

The system consists of 5 classes: *globalSeed* (described in section 4.2) *heaterController*, *heater*, *outsideWeather* and *heatedRoom*. There is only one object of each class.

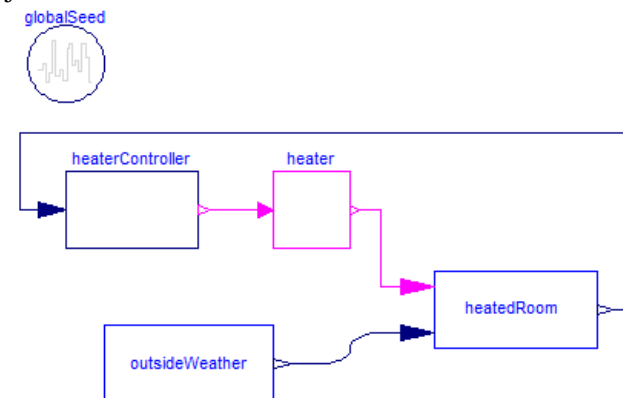


Figure 5. Structure of the Modelica model

In this case we can have a causal model, where each box has outputs calculated from its inputs. This is why the links are all directed.

The box *outsideWeather* just contains a constant parameter: the outside temperature, set to 17°C; this value is sent to *heatedRoom*. In *heatedRoom* there is the differential equation:

```
equation
  der(T) = 0.1 * (Outside_Temperature - T)
          + 5 * Heater_is_on;
```

The box *heaterController* just contains a hysteresis taken from the Modelica standard library. The two bounds (minimal and maximal temperatures respectively set to 15 and 20) are defined in this box. The output is a Boolean sent to the box *heater*; this Boolean is true whenever the heater is supposed to heat.

### 6.1 Heater model with stochastic transitions

With the method sketched in section 4, the heater is modelled as a continuous-time state machine where stochastic delay blocks with constant hazard rates are used inside the states (Figure 6).

For example “working” is the state when the heater is working. Inside this state a vector of delay blocks with fixed hazard rates are defined and the hazard rates are displayed in the icon ( $\{0.01\}/h$ ), so defining here 0.01 failures per hour. This state is instantiated with one delay block, and the transition from “working” to “notWorking” is just referencing the output of this delay block (`working.delay[1].y`).

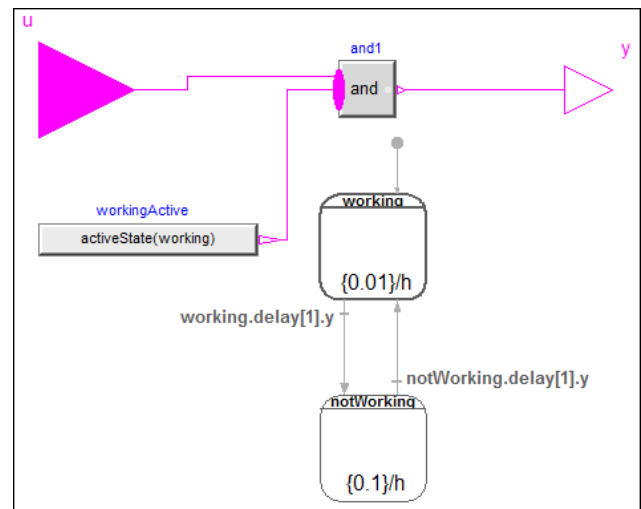


Figure 6: The heater model with a continuous-time state machine. In every state an instance of a delay block with constant hazard rate is present. In the transitions, the outputs of these delay blocks trigger the firing of the transitions.

This output becomes true after the stochastic delay defined by the hazard rate (a random number is drawn when state “working” is entered). In this case the state machine switches to “notWorking”. The repair rate is defined as 0.1 failures/h and the state switches back to “working” again, after this stochastic delay. The output of the heater model is true, if the state machine is in state “working” and the input of the heater (the signal coming from the controller) is true. The result of one simulation of the overall system is shown in figure 7.

## 6.2 Monte Carlo Simulation

The above model can directly be used to simulate *one realization* of the random process corresponding to the life of the system. To perform a Monte Carlo simulation to estimate, for example, the mean temperature as a function of time, it is necessary to generate a large number of such trajectories using different initial seeds for every simulation. This task can be performed in Dymola by using appropriate script functions (that are based on the algorithmic part of the Modelica language). A special Modelica/Dymola script has been implemented for this case to run the simulations and store the desired fractiles<sup>5</sup>. In figure 8, the mean value of the room temperature is shown, as well as the 1% and 99% fractiles at each time point respectively. 10 000 simulations were performed with 500 output points per simulation. On a notebook, these simulations took 25s. Computing the result values for figure 8 took another 45s (the reason is that a very simple algorithm was implemented to compute the fractiles and a better implementation will give a considerable speed-up).

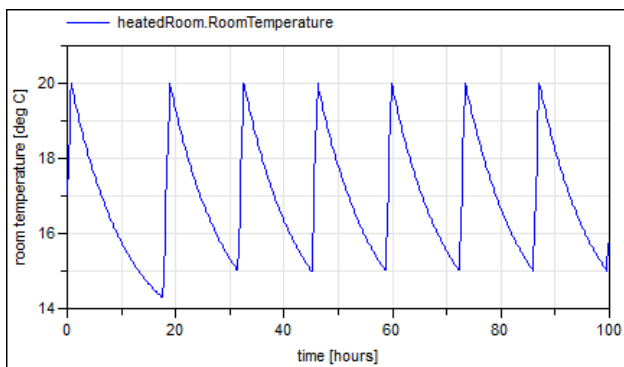


Figure 7: A single random trajectory of the temperature (containing one failure and one repair over 100 hours).

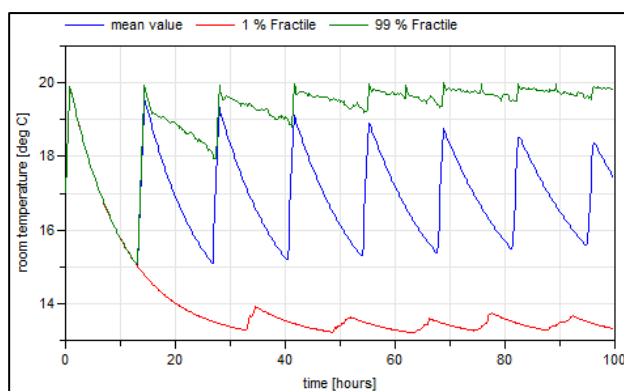


Figure 8: Statistics obtained from 10000 trajectories

<sup>5</sup> The computation of the 99 % fractile  $z$  from 10000 simulation runs means that at every grid point of the result 10000 result points are available and that 99 % of these are smaller than  $z$ . In other words, 1 % of 10000 values = 100 values are larger than  $z$ .

## 7 Comparison with approaches in the literature

The same problem (heated room) had been already modelled and solved with three other tools. The details about these experiments can be found in the two ESREL papers already mentioned.

The tool Vensim, which is well known in the domain of so called "system dynamics" was originally designed for modelling deterministic differential equations. Its graphical input interface is extremely limited, does not allow encapsulation of models, nor reuse by another means than copy-paste. This tool is not usable for real size systems, the building of models being much too error prone.

The tool KB3, based on the Figaro modeling language, is dedicated to the construction of discrete state stochastic models, for reliability and dependability calculations. With this tool it has been very easy to build a graphical model representing the heated room, using a library for hybrid stochastic Petri nets. The model was solved using the YAMS Monte Carlo simulator, able to process any Figaro model. The main concern with this approach is the impossibility to "separate" in the processing the calculations on the discrete and on the continuous part of the model. Thus it would probably be inefficient in terms of CPU consumption on a large model, just like the approach described in (Zhang et al. 2013), commented 15 lines below).

Finally, the tool PyCATSHOO based on Python libraries had also been tested. This tool is new and has no graphical interface. However, it uses an object oriented approach such as Modelica, with two distinct hierarchies, corresponding to the relations "is included in" and "inherits from". The PyCATSHOO models include a native notion of stochastic transition, since this tool was designed specifically for solving dynamic reliability problems. Its scalability is ensured by the use of state of the art libraries for solving differential equations and parallelisation of computations.

It will be interesting to make further comparisons of Modelica and PyCATSHOO models resolutions for larger systems.

In (Aldemir 1991) a more complex benchmark is described. From the solutions to this well-known benchmark (already mentioned in section 5.1), the one from (Zhang et al. 2013) is interesting, because here a general purpose continuous-time modeling environment (Simulink) is used together with a state machine (Stateflow) to solve a problem with a state dependent hazard rate. However in this reference, a fixed time step integration is utilized for the simulation and in every step the approximation is used that

the hazard rate is constant. The constant hazard rate is integrated over every time step and at the end of every time step random numbers are drawn to check whether the failures of components are to be triggered in the next time step, according to the current values of components failure rates. This technique is less precise and requires a lot more computations compared to the one explained in (Zhang and al. 2008). This *seems to be* the price to pay in order to be able to use high level models with a small modeling effort instead of ad hoc programs developed at a high cost in terms of manpower. But in fact, our new approach from section 3.2 is *an improvement* of the simulation strategy given in (Zhang and al. 2008), *without backtracking* and able to use error controlled, variable step-size integrators for the continuous part of the system and yet *it allows the use of high level models*, as usual in a Modelica environment.

## 8 Conclusions

In this paper we have pinpointed the need for considering probabilistic safety analyses in which the fault occurrence and propagation behavior can depend on the physical and control state of the considered system. Indeed, we advocated that reliability modeling should not be kept separate from modeling of physics and control, as it is today. One specific subtlety of the subject is that the joint consideration of reliability and physics require being able to consider state dependent hazard rates for time to failure distributions. As a first contribution, we provided an on-line procedure for Monte-Carlo simulation of such phenomena in the presence of coupling between fault events and system physics. Then, we proposed an extension of the Modelica language to support this kind of modeling. The extension has been meant minimal in that it most possibly relies on existing features. Not all probabilistic phenomena relevant to the joint simulation of reliability and physics are covered by our proposal, but we believe it is a first and significant contribution addressing a large part of the remaining open issues with current approaches.

## 9 Acknowledgements

This article is based on work carried out within the ITEA2 project MODRIO. Partial financial support of The German BMBF, the French DGCIS, and the Swedish VINNOVA for this development are highly appreciated.

## References

- Aldemir T (1991): **Utilization of the Cell-to-Cell Mapping Technique to Construct Markov Failure Models for Process Control Systems**. PSAM Proceedings, Elsevier Publishing Company Co. Inc., NY, pp 1431-1436.
- Bouissou M., Jankovic M. (2012): **Critical comparison of two user friendly tools to study Piecewise Deterministic Markov Processes (PDMP)**. ESREL 2012, Helsinki.
- Bouissou M., Chraïbi H., Chubarova I. (2013): **Critical comparison of two user friendly tools to study Piecewise Deterministic Markov Processes (PDMP): season 2**. ESREL 2013, Amsterdam.
- Davis M.H.A (1993): **Markov Models and Optimization**, Chapman & Hall
- Elmqvist H., Gaucher F., Mattsson S.E., Dupont F. (2012): **State Machines in Modelica**. Modelica'2012 Conference, Munich, Germany, Sept. 3-5, 2012. Download: <http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf>
- Elmqvist H., Mattsson S.E., Otter M. (2014): **Modelica extensions for multi-mode DAE systems**. Modelica'2014 Conference, Lund, Sweden, March 10-12.
- Looye G., Joos H.D. (2006): **Design of Autoland Controller Functions with Multi-objective Optimization**. Journal of Guidance, Control, and Dynamics, Vol. 29, No. 2, March-April, pp. 475 -484.
- Marseguerra M. and E. Zio (1996): **Monte Carlo Approach to PSA for dynamic process system**. Reliability Engineering and System Safety, 52:227-241.
- Sheldon M.R. (1990): **A course in simulation**. Mc Millan, ISBN 0-02-403891-1
- Tuffin B., D. S. Chen, and K. Trivedi (2001): **Comparison of hybrid systems and fluid stochastic Petri nets**. Discrete Event Dynamic Systems, 11 (1/2):77-95.
- Zhang H., Dufour F., Dutuit Y., and Gonzalez K. (2008): **Piecewise deterministic Markov processes and dynamic reliability**. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability 222(4), 545-551.
- Zhang H., Saporta B., Dufoura F., and Deleuzed G. (2013): **Dynamic Reliability by Using Simulink and Stateflow**. Chemical Engineering Transactions. Vol. 33, pp. 529-534



# The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft

Andreas Klöckner

German Aerospace Center (DLR), Institute of System Dynamics and Control  
82234 Weßling, Germany, [Andreas.Kloeckner@dlr.de](mailto:Andreas.Kloeckner@dlr.de)

## Abstract

The paper introduces a continuous-time architecture and a Modelica library for mission planning based on behavior trees. It allows to study the long-time behavior of complex aircraft models in interaction with reactive mission plans by means of efficient simulations. The developed Modelica library is used in a mission example for a solar high-altitude aircraft and the advantages of the behavior tree formulation in both simulation speed and modularity are discussed. The architecture will further be used to deploy automatically coded mission plans to actual flight computers using the functional mockup interface.

*Keywords:* UAV; Mission Management; Behavior Trees; Autonomy; Artificial Intelligence

## 1 Introduction

Missions currently envisaged for Unmanned Aerial Systems (UAS) pose increasing demands on modeling, planning and simulation capabilities. For example, solar UAS are highly dependent on environmental conditions and must execute autonomous missions lasting several weeks (see Fig. 1).

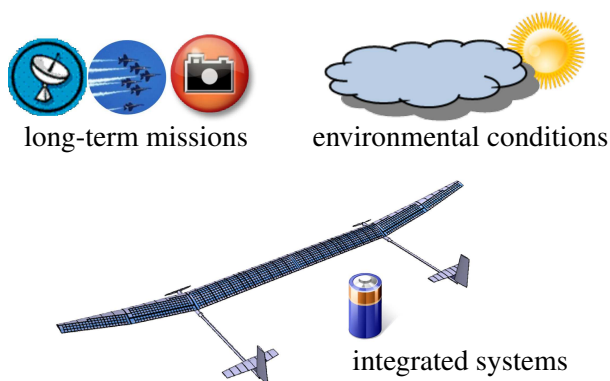


Figure 1: The growing complexity of integrated unmanned aircraft systems, environmental conditions and long-term missions call for efficient and intuitive tools for mission design and simulation.

With growing complexity of the systems, integrated simulation schemes have to be employed, not only modeling the vehicle's flight dynamics but also its avionics systems or even aeroelasticity. Such models have to be as detailed as possible, while maintaining sufficient simulation speed to also run simulations of long-term missions. We have shown previously that Modelica is an excellent tool for this purpose [1].

Additionally, missions of increased complexity and diversity make use of a range of distinctly developed UAS capabilities such as collision avoidance, formation flying or physical interaction with the environment. In order to further improve a system's versatility, developers seek to employ the same system for a variety of purposes, providing as much autonomy to the system as possible (see e.g. [2]).

In order to face these challenges, a flexible, scalable and intuitive scheme for UAS control systems and mission plans has to be provided. Ögren recently proposed behavior trees for this purpose [3]. They are distinguished by their standardized structure providing a mission design scheme, which Champandard argues to combine important advantages of different schemes such as state machines and task planners [4].

However, conventional behavior tree implementations rely on periodically updating the tree's status based on its inputs. This *clocked* or *discrete-time* processing makes previous behavior tree formulations unsuitable for continuous-time simulation of long-term missions. The tree's update rate would have to be chosen high enough to correctly reflect the system's behavior. The smaller integrator time-steps and additional time events would in turn slow down the overall simulation speed considerably.

In order to combine efficient long-term simulations with the capabilities of behavior tree mission plans, a *continuous-time* formulation for behavior trees was thus developed and implemented in Modelica. The present paper describes this formulation and the resulting Modelica library:

- The conventional behavior tree formulation is characterized by discrete-time, sequential, top-down processing of the tree. The new scheme is laid out as continuous-time, event-driven, and bottom-up processing, see Sec. 3. This allows a simulator to choose large integration step-sizes as desired for long-term mission simulation. The formulation can be generalized to other languages supporting event notifications.
- Section 4 introduces the Modelica implementation of the modified system. A library of base tasks with clear internal and external interfaces allows the user to graphically design mission plans and also easily implement new task types. Additional infrastructure is provided to simplify communication with the tree. The processing of an exemplary simple task is shown in Sec. 5.
- The example shown in Sec. 6 show-cases the modular assembly of a mission plan for a solar UAS. The simulation speed of the controlled system is maintained. A comparison to a state-machine based approach using the StateGraph2 library [5] gives identical results.

## 2 Behavior Trees

Behavior Trees are a technique developed for artificial intelligence in computer games. They are first mentioned in this context by Damian Isla [6]. A good introduction is found in Millington's textbook [7]. The approach is introduced to the UAS world by Ögren [3].

Using behavior trees, complex missions are built up using atomic tasks. These can e.g. query the aircraft's state (conditions) or send commands to the underlying control system (actions). A typical combination of the tasks is to conditionally execute an action. A solar UAS e.g. might need to harvest solar energy by maximizing its flight altitude, if a surplus of energy is available. In behavior trees, this is expressed using a sequence of sub-tasks as shown in Fig. 2a.

The task of maximizing potential energy could then be further composed of two alternatives: Either the maximum altitude is already reached or the aircraft has to climb. A set of alternative approaches to a common goal is expressed using a selector as shown in Fig. 2b.

A sequence executes all its sub-tasks in the given order until all sub-tasks are finished successfully. The selector also executes its sub-tasks in the given order, but returns successfully with one successfully finished sub-task. The two basic composite tasks can thus be compared to logical AND and OR operators.

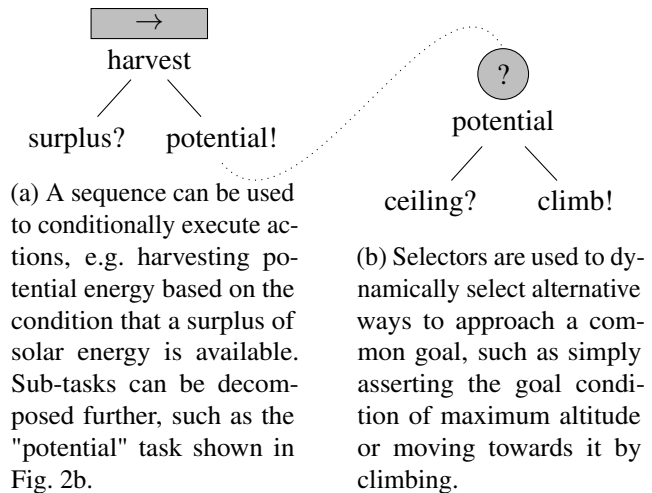


Figure 2: In a behavior tree, simple tasks are connected to a tree in order to achieve more complex goals.

The main advantage of behavior trees for UAS mission management is their standardized and intuitive structure. All atomic tasks describe self-contained and goal-directed behaviors. Higher-level plans are built by intuitively decomposing complex goals into sub-goals. Since all tasks have a common interface and the switching of tasks is determined by implicit logics, the user can modularly interchange arbitrary tasks in the tree. The plans are thus very scalable and human-readable on all levels of the hierarchy. Additionally, behavior trees are inherently memory-free and work with persistent statuses. They thus execute the correct task immediately after a restart or online modification.

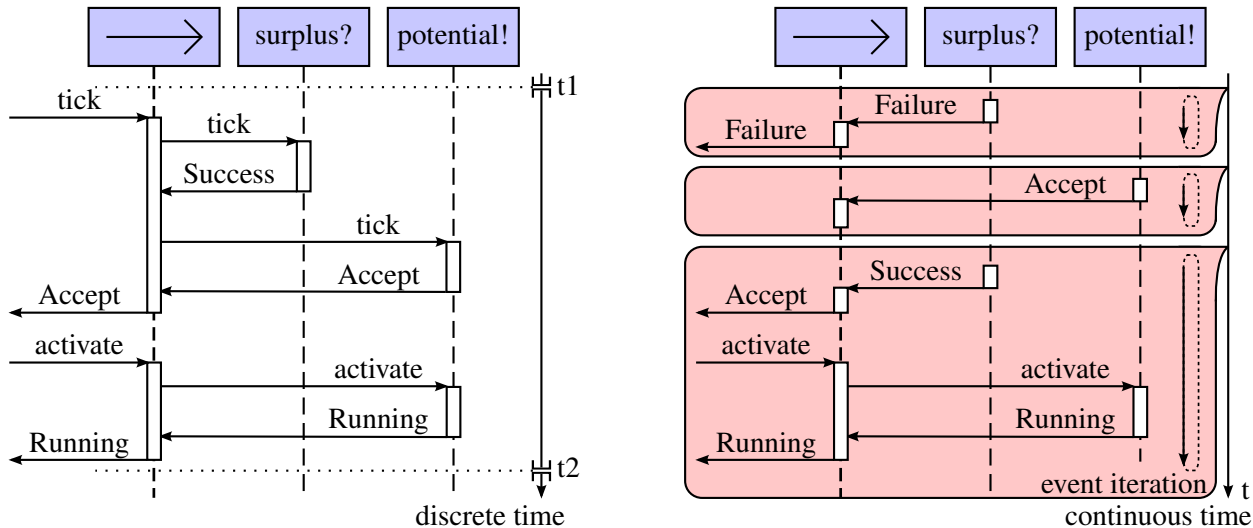
In previous studies, I have augmented behavior trees with state-machine-like entry and exit hooks [8]. The necessary notion of transient tasks is complemented in that paper with an introduction to the processing of basic behavior trees and further discussion of advantages as well as prospects of behavior trees for UAS mission management. A first step towards logical verification and validation of behavior trees is also covered by previous research [9]. Here, the formalism is refined for continuous system simulation.

## 3 Continuous-time Processing

Conventional behavior trees are processed at a fixed update rate. At each instance of time, a tick is issued to the top-most node of the tree. A tick requests a task to report its status back to the superior task. The standard statuses of a task are Success, Failure and Running.

The tick is propagated in a top-down manner through the tree to its leaf nodes. Each composite task, such as a sequence or a selector, sequentially ticks its sub-tasks to determine their statuses. Figure 3a uses a sequence





(a) In conventional processing, a tick is propagated from the root node of the tree down to the leaf tasks. After all sub-tasks have been ticked, a task returns its own status to its superior task. If necessary, additional activation routines are executed subsequently. This processing is done in each discrete time step.

(b) The proposed continuous-time processing relies on the sub-tasks notifying their superior tasks of status changes. In Modelica, these notifications are generated in event iterations and passed up the tree. Activation procedures are still triggered from the top. This processing can be done in continuous-time.

Figure 3: The processing of tasks in behavior trees is conventionally done sequentially in discrete time. In the new processing scheme, status notifications are passed through the tree in continuous time. Both variants are contrasted here in form of sequence diagrams showing processing of the example sequence from Fig. 2a.

diagram to illustrate the process of ticking the example sequence from Fig. 2a. The sequence is not running at first. In the illustrated time-step, it asserts a surplus of solar energy and then decides to accept being activated. It is subsequently activated by its superior node, and then activates its harvesting sub-task in turn. The separate decision and activation parts of the process are a consequence of the activation/deactivation procedures introduced previously [8].

The downside of this straight-forward processing is that a tick must be issued to the root node with a sufficiently high repetition rate in order to react adequately to changes in the environment. On the Modelica side, this is solved using discrete-time events. These in turn slow down long-term continuous simulations, which cannot make use of large integrator step-sizes anymore and have to handle additional integrator restarts.

In order to overcome this downside, I introduce the bottom-up approach for the decision part of the processing shown in Fig. 3b. Instead of periodically querying the sub-tasks' statuses, the superior task is notified by its sub-tasks about status changes. The superior task may then adapt its own status to the new situation and propagate the new status towards the root of the tree. The activation part of the processing is not changed.

Using this scheme, the example sequence starts with a Failure status, because it cannot assert a surplus of solar energy. The harvesting sub-task can then asynchronously decide to accept being activated. This does not require any status change of the sequence, such that the change event is not passed up the tree. Finally, the surplus sub-task notifies the sequence of its new Success status. The sequence then passes on its Accept status to the higher levels of the tree and is activated as in conventional processing. Each of these status notifications is processed in an event iteration.

If the behavior tree processing is changed in the proposed way, it is relieved of most of the discrete-time events. Instead, state events will be triggered in Modelica whenever a task changes its status. This allows for continuous-time simulation. Additionally, status changes of the behavior tree are resolved instantly as if the tree had an infinitely high update rate. The additional cost of iterating state events is moderate, because typical behavior tree mission plans change their status at a much larger time-scale than the simulation of the controlled system.

The new processing scheme additionally addresses the known limitation of behavior trees in handling events like a finite state machine as pointed out in pre-

vious work [8]. With the presented processing, the foundations are laid to implement event-based behavior trees in any programming language supporting event notifications. Additional work will have to be spent on adding internal or external storage to the tree in order to convert instantaneous events to permanent statuses.

## 4 Modelica Implementation

In Modelica, all tasks are implemented as models extending from a super-class Task, in which the common properties of all tasks are defined. The Task model in particular implements the status switching logics of the tasks. The status cycle used in this work is an extended version of the one introduced previously [8]. It makes a distinction between transient tasks and non-transient tasks. Transient tasks such as conditions can be evaluated without activation. Non-transient tasks such as an action must be activated in order to evaluate them.

All possible statuses of a task are defined in a Status enumeration. It defines the following eight statuses:

**Success and Failure** are the standard transient statuses and indicate, if the task (usually a combination of conditions) evaluates successfully or not.

**Accept** indicates that the task will accept being activated. It marks the entry point to the non-transient part of the status cycle. A task can only enter the non-transient part of the status cycle, if it is activated by its superior task. This status was called Activating in [8].

**Activating** now designates a task, which was properly activated by its superior task and is currently performing initialization procedures. This status can be used to execute the entry hook of the task. It is introduced here in order to allow for time-consuming initialization procedures. These were not possible previously.

**Running** denotes a properly initialized, non-transient task during its nominal execution.

**Finished and Aborted** are the non-transient equivalent statuses to Success and Failure. They indicate, if a task has finished running successfully, or if it failed during its execution. These statuses mark the exit point from the non-transient part of the status cycle. A task can only leave these statuses, if it is deactivated by its superior task.

**Deactivating**, equivalently to Activating, designates a task, which was deactivated by its superior task and is currently performing finalization procedures. It can be used to run the task's exit hook.

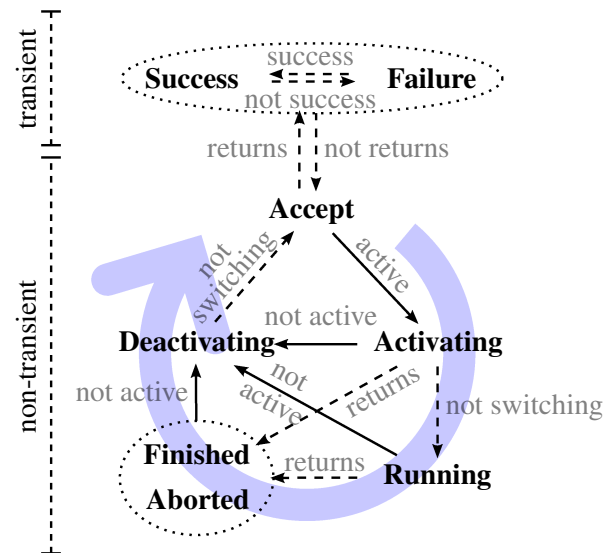


Figure 4: The status of a task is triggered by the internal trigger flags returns, success and switching ( - - ) or by (de-)activation through the active flag ( — ).

Figure 4 summarizes the complete status cycle. The task may internally determine status changes marked with dashed arrows. The status changes marked with solid arrows can only be initiated by the superior task through activation or deactivation.

Until the task is activated it can choose its status freely from Success, Failure and Accept. This allows transient tasks, especially conditions, to be evaluated without activation. As soon as the task enters the non-transient part of the status cycle by being activated, it is basically bound to iterate its status in the marked clockwise direction. The task can finally leave the non-transient status cycle only if deactivated by its superior task and after passing the Deactivating status.

In order to relieve task designers from coping with the correct implementation of the status cycle, three additional trigger flags are introduced. These indirectly trigger a task's status:

**returns** determines, if the task can be evaluated in the sense of returning a successful or unsuccessful status or if the task needs to be activated first.

**success** determines the correct status, if the task has a returning status.

**switching** is used to mark initialization and finalization of tasks. It can be used e.g. while powering up resources needed by the task.

The task's status can now be uniquely determined using these trigger flags and an additional active flag provided by the superior task. The trigger flags are also illustrated in Fig. 4 by the labels of the arrows.

The appropriate equation is common to all tasks and a base Task model is thus provided (see Listing 1). It contains the status switch logics, the three internal trigger flags returns, success and switching as well as a public uplink connector to the superior task. In this way, the status dynamics of all tasks are forced to be consistent and new task types can be conceived by simply assigning values to the trigger flags.

Listing 1: The basic Task interface contains an uplink to the superior task, three additional status triggers and the status switch logics. The function `f()` implements the status cycle shown in Fig. 4.

```

partial model Task
  Uplink uplink;
protected
  Status status = uplink.status;
  Boolean active = uplink.active;
  Boolean returns;
  Boolean success;
  Boolean switching;
equation
  status = f(pre(status),
            active,
            returns, success, switching);
end Task;

```

A Condition is e.g. defined by Listing 2. It can always be evaluated to Success or Failure and thus fixes the returns flag to true. The success flag determines whether to return Success or Failure. It can be filled by arbitrary conditional equations, e.g. using an additional BooleanInput. Because a condition cannot be activated, the switching flag must actually only be filled in order to balance the model.

Listing 2: Defining a new Condition task is done by binding the trigger flags to boolean expressions.

```

model Condition
  extends Task;
  BooleanInput u      "A boolean input";
equation
  success = u;      //is used as condition
  returns = true;  //and always returned
  switching= false; // => Not used!
end Condition;

```

All inter-task communication described in Fig. 3 is handled by the Task's UpLink connector and corresponding DownLink connectors of the superior tasks. The UpLink connector is outlined in Listing 3. The status variable carries the status information, while the active flag is used by the composite tasks to activate or deactivate their sub-tasks. Using these connec-

tors as public interfaces, single tasks can be connected in a tree to almost arbitrarily complex mission plans.

Listing 3: The BehaviorTrees UpLink connector for connecting sub-tasks to their respective superior task passes the sub-task's status up the tree and receives an active flag from the superior task.

```

connector UpLink
  output Status status "The task status";
  input Boolean active "(De-)Activation";
end UpLink;

```

Using the interfaces described above, a library of basic tasks is provided. The library's structure is shown in Fig. 5. It includes the most common tasks encountered in behavior trees such as Condition, Action, Selector and Sequence. The Root task is needed to steer the overall execution of the tree.

Additionally, a communication structure is provided in order to simplify the information flow between the tree and its environment (Blackboard). The system relies on a global memory block using Modelica's inner/outer functionality. Continuous-time Real variables can be written to the system using the SetReal block. Each slot is provided with a name and an active flag. The GetReal block retrieves the currently active value with a given name from the blackboard. Using this structure, it is possible for several tasks to write to the same input of the controlled system. It especially allows to also encapsulate continuous-time controllers in tasks and to switch between them automatically based on the current behavior tree status.

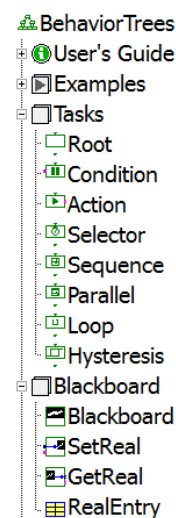


Figure 5: The BehaviorTrees library contains a number of standard task implementations. It also provides infrastructure to facilitate communication with the tree.

## 5 Processing of a Sequence

In this section, the processing of the last event shown in Fig. 3b is detailed with respect to the implementation described in the previous section. The Sequence task uses the implementation shown as pseudo-code in Listing 4. It first processes the sub-tasks' statuses to determine its own status triggers. These can then be processed by the standardized status cycle implementation as shown in Fig. 4. If the sequence is active, it passes on this flag to its first accepting sub-task.

Listing 4: The Modelica pseudo-code for the implementation of a Sequence Task determines the status triggers and steers the active flag of its sub-tasks.

```

model Sequence
  extends Composite "Task with sub-tasks";
  equation
    returns      = not any substatus[:] is
                  Accept, Activating, Running;
    success      = all substatus[:] are
                  Success or Finished;
    switching     = any substatus[:] is
                  Activating, Deactivating;
  // <-- Apply status cycle from Task
  if active then
    subactive[first accepting] = true;
  end if;
end Sequence;
    
```

Initially, the surplus sub-task has the status Failure and the potential sub-task has the status Accept. The sequence consistently returns Failure.

When the surplus status changes to Success, the sequence's returns flag changes to false. Applying the status cycle, the sequence changes its status to Accept and passes on this status to its superior task.

According to the switch logics contained in the superior tasks, the sequence's active trigger is then changed to true. The sequence passes on this value to its first accepting sub-task, the potential energy harvesting. This action then changes its status to Activating, which in turn changes the sequence's switching trigger to true. According to the status cycle, the sequence also assumes Activating status and passes it up the tree.

Eventually, the potential sub-task generates a new event, when it switches its status to Running. The sequence adapts to this change by changing its own switching flag to false, applying the status cycle, and passing its new Running status to its superior task.

Figure 6 shows a sequence diagram of this sequence of events. The single final event from Fig. 3b is split in three events here. Depending on the superior tasks

and the sub-tasks, these events can also happen immediately one after the other. An action without activation procedure e.g. immediately leaves the Activating status. This combines the latter two events into one.

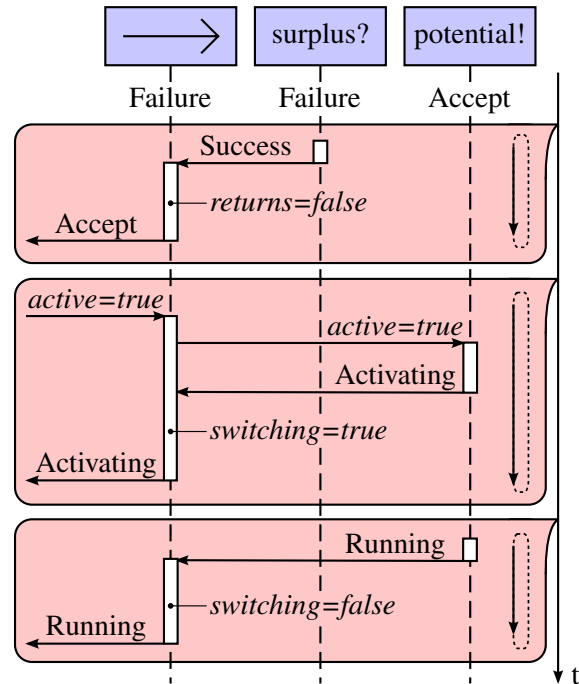


Figure 6: The detailed processing of a sequence activating its potential sub-task includes the status triggers.

## 6 Application Example

The described system is used for show-casing its applicability to actual UAS mission scenarios. To this end, a mission plan is built for an electric high-altitude solar-powered aircraft. This plan is used to steer the integrated simulation model developed in previous work [1]. The aircraft is implemented as the non-linear inverse of a point-mass model of the flight dynamics with two controlled propellers. It has 26 individually controlled solar panels. The total number of continuous-time states for the aircraft is 72.

The plan is divided in a longitudinal and a lateral part. The lateral part steers the aircraft towards a holding position and commands a holding pattern at this point. This part is not described here in detail. The longitudinal part of the plan is shown in Fig. 7. It consists of two sub-goals. The first is to ensure that a maximum of solar energy is stored in the aircraft. The second includes the main mission, such as operating a communication relay at the holding position. In this example, it is simplified by issuing an altitude holding command.

The energy-maximizing plan is comprised of two alternative plans. One is used to harvest solar energy. It

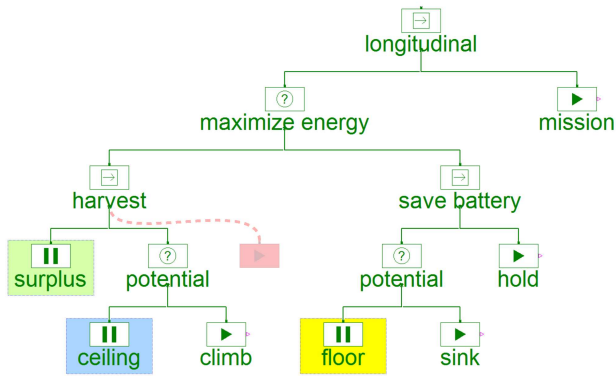


Figure 7: The BehaviorTrees mission plan uses encapsulated inputs and outputs specialized for the specific UAS application. The conditions are marked with colored frames for comparison with the transitions in Fig. 8. The plan's modularity is shown by introducing a new sub-task of the harvest task (shaded, dashed line). Only one additional connection is required for this extension.

is described already in Sec. 2 as the introductory example. It commands the aircraft to climb to its maximum altitude, whenever a surplus of solar energy is available. The second alternative ensures that a minimum of energy is used, if no energy can be stored. To this end, the aircraft first descends to a minimum altitude and holds this altitude in a way, which minimizes battery use. This type of using altitude to store potential energy is called a "jojo" mission. It is important to notice, that in this plan formulation, the main mission is only allowed to be executed, if no energy task of higher priority needs to be executed.

Figure 8 shows a StateGraph2 [5] implementation equivalent to the behavior tree. It consists of the four states *mission*, *climb*, *sink* and *hold* as well as numerous transitions with conditions equivalent to the behavior tree logic. The corresponding state graph transitions and behavior tree conditions are marked with colored frames in Fig. 7 and 8 respectively. It can be noted, that some transitions have to be reused multiple times in order to replicate the behavior tree's results.

The advanced modularity of the behavior tree can be appreciated even more, when an additional task is introduced. Figures 7 and 8 also include a sketch of loading a fuel cell as an additional energy harvesting task. While the behavior tree only requires one additional connection, the state graph has to be extended with a number of transitions. The extension is done here in an ad-hoc way and optimizations of the state machine are possible. However, the typical mission

designer will appreciate the facilities to create ad-hoc plan changes in an efficient way such as provided by the BehaviorTrees library.

The results of both the BehaviorTrees and the StateGraph2 mission plans are shown in Fig. 9. A full day of flight (86400 s) is shown including a full jojo mission between 6000 m and 13000 m of altitude. The results of both mission plans are identical: The simulation starts before sunrise, such that the aircraft holds its lower altitude limit. When the sun rises at about 07:50, the batteries start charging. Only when the batteries are fully charged at about 11:30, the aircraft climbs to its mission altitude. With decreasing solar energy in the early evening, the aircraft starts to sink to its lower altitude limit again.

Figure 9 also shows the related current productions. The repeated switching with critical battery stems from highly different power demands for the altitude holding and sinking tasks in combination with discontinuous commands from the mission plan. Providing fully continuous switching will alleviate this behavior.

In order to compare the computational complexity of the different implementations, the required CPU time on a standard laptop computer and the generated number of time- and state-events are given in Tab. 1. The values are compared for the discrete-time formulation of the behavior tree and the continuous-time BehaviorTrees modification. The tree is ticked once per minute in the discrete-time case. The important reduction of both time events and execution time can be seen. The continuous-time formulation saves about 80 % of the simulation time and removes all time events. For completeness, the execution times of the StateGraph2 implementation are also given, as well as the results from simulating only the aircraft model with the recorded command inputs for the same period.

Table 1: The execution measures of the different configurations are determined for a single simulation experiment of the discrete-time and the continuous-time behavior tree as well as the state graph implementation and a direct simulation with recorded inputs. CPU time, number of time events and number of state events are extracted from the simulation log file.

Configuration	CPU time	time- / state events
Discrete	368 s	1442 249
Continuous	72 s	0 247
State graph	76 s	60 242
Direct inputs	66 s	25 183

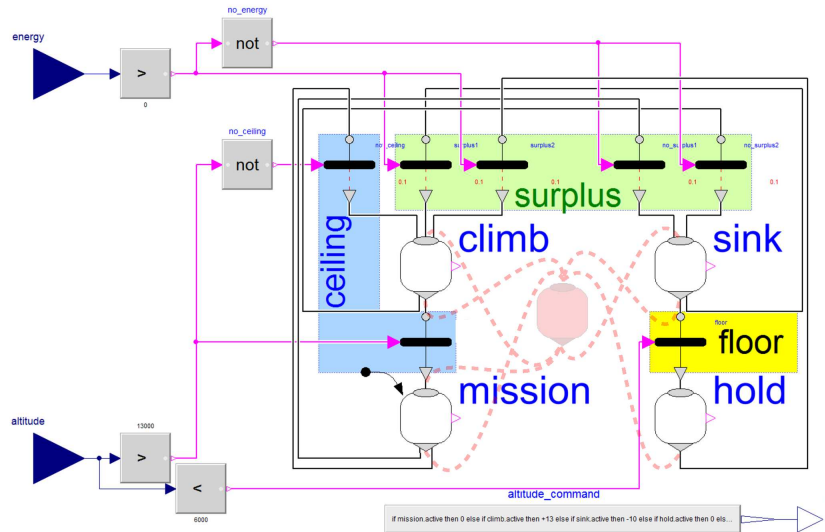
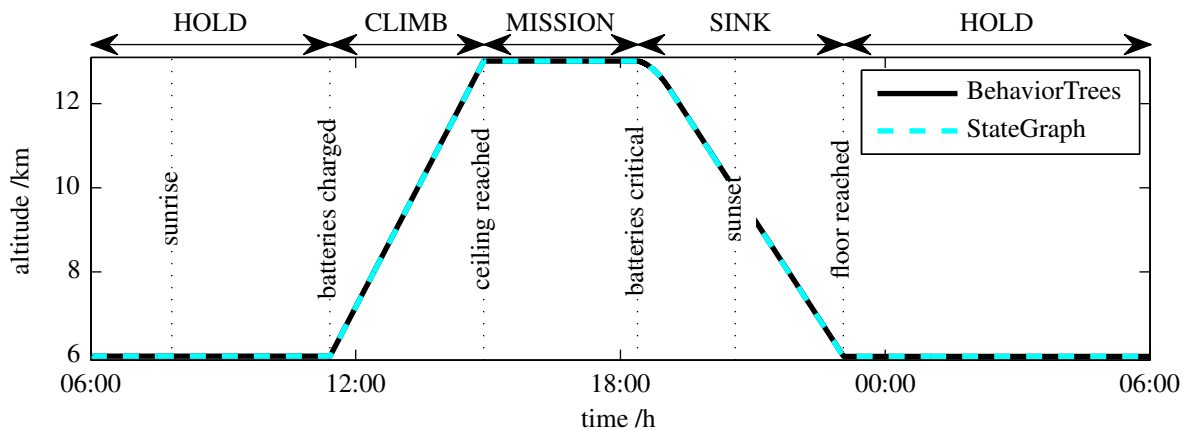
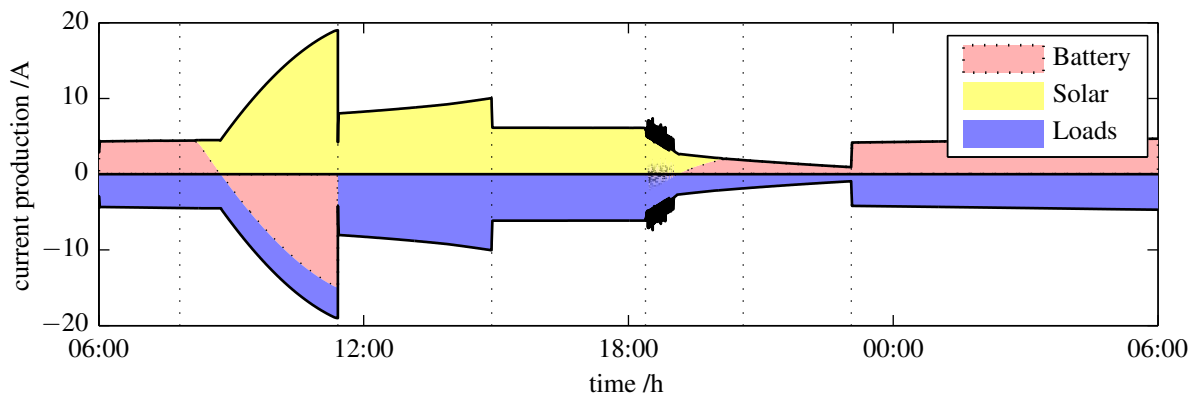


Figure 8: The StateGraph2 mission plan consists of four places corresponding to the actions of the BehaviorTrees mission plan. The conditions of the behavior tree are translated into transitions. They are marked with colored frames for comparison with Fig. 7. It can be seen that the state machine implementation requires a number of redundant transitions. The complexity of this implementation becomes more visible, when an additional task is introduced similarly to the one included in Fig. 7 (shaded, dashed lines).



(a) The altitude results for a full day of flight are identical for both mission plans.



(b) Also, the energy production and consumption patterns are equal for both simulations.

Figure 9: Both mission plans are simulated with an integrated aircraft model for a full day of flight. The results are identical for both plans.

## 7 Conclusions and Outlook

The conventional discrete-time formulation of behavior trees can be argued unsuitable for long-term UAS mission simulation in Modelica because of its forced update rate. A continuous-time formulation is thus devised and implemented in the Modelica BehaviorTrees library. The formulation allows the simulator to choose large step-sizes, allowing for long-term mission simulation with behavior tree mission plans in Modelica. The approach relies on passing event notifications between composite tasks and their sub-tasks.

The new formulation also addresses the limitations of previous behavior tree formulations with respect to event-handling. It provides the capability needed to process behavior trees based on events also in languages other than Modelica. By providing a suitable data storage, the approach can be used to create event-driven behavior trees also in real-time environments such as flight computers.

The Modelica library described in this paper presents clear public and internal interfaces, allowing the user to design mission plans graphically on the one hand and to implement new task types on the text layer on the other hand. The behavior tree infrastructure is complemented with a communication infrastructure allowing to conveniently pass signals from and to the plan without direct connections.

A plan implemented with the BehaviorTrees library gives identical results compared to an equivalent plan using the StateGraph2 formalism. The simulation speeds confirm that long-term simulations are possible with both methods. However, the plan's modularity is increased using the BehaviorTrees facilities as compared to a StateGraph2 implementation.

The comparison of results and timings should be regarded as an initial qualitative result. It indicates that behavior trees can indeed conveniently be used to steer a solar-powered high-altitude aircraft on long-term missions. Quantitative evaluations and comparisons to conventional state machines and discrete real-time implementations still need to be carried out. Formal validation and verification need to be addressed in order to guarantee consistent behavior between the continuous and discrete formulations.

The BehaviorTrees library is currently only used in internal studies and research projects. A commercial, or open-source, release is currently not planned but not ruled out either. The library will be further developed in the scope of a doctoral project and interested users are invited to contact the author for further information.

Future versions of the library will make use of the new synchronous elements provided by the Modelica 3.3 specification. These should provide for even better performance and robustness. Comparisons to pure Modelica 3.3 state charts should give similar results as the comparison to StateGraph2: Similar performance with improved modularity. Additional improvements can be made with respect to repeated switching by providing for continuous switching, where possible.

In summary, the prospects of using BehaviorTrees mission plans in Modelica are excellent, both in usability and in performance. The approach can be valuable not only for UAS applications, but also for other fields such as vehicle test automation. Future research effort will additionally be spent on deploying such mission plans to actual flight computers using automatic code generation and the functional mockup interface.

## References

- [1] Andreas Klöckner, Martin Leitner, Daniel Schlabe, and Gertjan Looye. Integrated Modelling of an Unmanned High-Altitude Solar-Powered Aircraft for Control Law Design Analysis. In Qiping Chu, Bob Mulder, Daniel Choukroun, Erik-Jan van Kampen, Coen de Visser, and Gertjan Looye, editors, *Advances in Aerospace Guidance Navigation and Control – Selected Papers of the Second CEAS Specialist Conference on Guidance, Navigation and Control*, pages 535–548. Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-38252-9.
- [2] Maximilian Laiacker, Andreas Klöckner, Konstantin Kondak, Marc Schwarzbach, Gertjan Looye, Dominik Sommer, and Ingo Kossyk. Modular scalable system for operation and testing of UAVs. In *American Control Conference*, pages 1462–1467, Washington, DC, 17-19 June 2013. IEEE. ISBN: 978-1-4799-0176-0.
- [3] Petter Ögren. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. In *AIAA Guidance, Navigation and Control Conference*, Minneapolis, Minnesota, 13 - 16 August 2012. AIAA. AIAA 2012-4458.
- [4] Alex J. Champandard. Behavior Trees for Next-Gen Game AI. In *Game Developers Conference*, Lyon, France, 3-4 Decembre 2007.
- [5] Martin Otter, Martin Malmheden, Hilding Elmqvist, Sven Erik Mattson, and Charlotta Johnsson. A new formalism for modeling of reactive and hybrid systems. In *Proceedings of the 7th International Modelica Conference*, pages 364–377. Linköping University Electronic Press, 2009.
- [6] Damian Isla. Handling complexity in the Halo 2 AI. In *Game Developers Conference*, 2005.
- [7] Ian Millington and John Funge. *Artificial intelligence for games*. Morgan Kaufmann, Burlington, MA, 2nd edition, 2009. ISBN: 978-0123747310.

- [8] Andreas Klöckner. Behavior Trees for UAV Mission Management. In Matthias Horbach, editor, *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*, volume P-220 of *GI-Edition-Lecture Notes in Informatics (LNI) - Proceedings*, pages 57–68, Koblenz, Germany, 16-20 September 2013. Gesellschaft für Informatik e.V. (GI), Köllen Druck + Verlag GmbH, Bonn. ISBN 978-3-88579-614-5.
- [9] Andreas Klöckner. Interfacing Behavior Trees with the World Using Description Logic. In *AIAA Guidance, Navigation, and Control Conference*, Boston, MA, 19-22 August 2013. AIAA. AIAA 2013-4636. ISBN 978-1-62410-224-0.



# Multi-Level Library of Electrical Machines for Aerospace Applications

P. Giangrande      C.I. Hill      C. Gerada      S.V. Bozhko  
University of Nottingham  
PEMC Group, Nottingham, NG7 2RD, UK.  
p.giangrande@nottingham.ac.uk      c.hill@nottingham.ac.uk

## Abstract

An electrical machine library, developed within the framework of the European project *Actuation 2015*, is presented in this paper. The library has been developed adopting a multi-level approach, in order to minimize the models complexity and reduce the computational time. Multi-level approach consists in creating several models of the same electrical machine topology, with different levels of complexity. Indeed, model complexity increases at higher model levels and each model takes into account specific physical effects. In addition to the fundamental behavior, the presented models address physical effects such as losses, magnetic saturation, torque ripple and fault conditions. The interchangeability among model levels is ensured by using common interfaces. An overview of the library structure is given; however, particular attention is paid on the permanent magnet synchronous machine (PMSM) models, since they are becoming increasingly widespread in aerospace applications. The PMSM models description and simulation results are provided, in order to highlight the implemented physical effects and confirm the models effectiveness.

*Keywords: Permanent Magnet Synchronous Machine, Multi-Level, Losses, Magnetic Saturation, Torque Ripple, Fault conditions.*

## 1 Introduction

According to the concept of the More Electric Aircraft (MEA), an increasing number of hydraulic actuators for primary and secondary flight control surfaces are being replaced by Electromechanical Actuators (EMAs), in order to improve efficiency, weight and maintenance of the aircraft [1]. For this reason, EMAs are becoming an attractive research area and part of the research work is focused on developing virtual testing environments and analysis tools. Indeed, EMAs' manufacturers require simulation tools capable of analyzing EMAs performance and exploring different design configurations. In response to

these requirements, a suitable simulation and analysis tool for EMAs, named *Actuator* library, has been developed within the framework of the European project *Actuation 2015*.

The multidisciplinary architecture of EMAs could lead to a large simulation time or even numerical non-convergence due to the model complexity [2]. In order to avoid these issues, the idea of implementing several models of the same component (i.e.: inverter, electrical machine, gear box etc...), with different levels of complexity has been exploited. Hence, a multi-level approach has been adopted, with the aim of keeping the models as simple as possible, since a good model is a wise trade-off between realism and simplicity.

The model levels are categorized by the complexity of the physical effects implemented within the model. In particular, higher model levels take into account more complex physical effects. Considering that several models of the same component are available within the library, the interchangeability among the model levels is crucial, in order to investigate different physical effects by simply replacing the model level. For this reason, the interchangeability has been ensured by using common interfaces among the model levels.

In this paper, the *Electrical Machines* library, which is part of the *Actuator* library, is presented. The *Electrical Machines* library has been implemented using Modelica [3], as modelling language, and Dymola, as simulation environment. This library is organized as three packages and each of them considers a specific machine topology, such as PMSM, synchronous reluctance machine (SRM) and direct current (DC) machine. Each one of these packages contains the sub-package *Examples*, which provides several case studies, helpful for highlighting the features of each modelling level. Due to space constraints, not all the machine topologies included in the *Electrical Machines* library will be considered in this paper. Indeed, the presented work is mainly focused on describing the

PMSM models as these machines are characterized by an excellent efficiency, together with a high power density [4]. These features make PMSMs very attractive for aerospace applications [5]. Whilst initially resistance to the adoption of these machines was shown by airframes to the possibility of unsuppressed short-circuit currents, it is now more accepted that through fault tolerant design and health monitoring schemes these machines offer an optimum solution.

In the next sections, an overview of the `Electrical Machines` library is presented along with a description of the model interfaces. Moreover, details of the PMSM modelling levels and the physical effects (i.e.: losses, magnetic saturation, torque ripple and fault conditions), taken into account, are given. Simulation results obtained using Dymola are also included.

## 2 Library Structure

An overview of the `Electrical Machines` library is provided in this section and figure 1 shows the library structure.

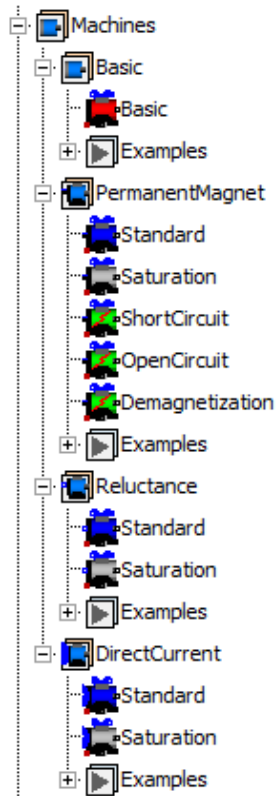


Figure 1: Structure of the `Electrical Machines` library as part of the `Actuator` library.

The machines considered in this library are PMSM, SRM and DC machine. The PMSM type covers a number of topologies from non-salient surface mount to high reluctance interior permanent magnet (IPM) machines. The library is organized in three packages,

one for each machine topology earlier mentioned, plus the package `Basic`, which considers a generic alternating current (AC) electrical machine. According to the multi-level approach previously introduced, several models with different level of complexity are included within each of the four packages. The level of complexity is related with the physical effects that the model takes into account. In other words, a higher level of complexity implies the modelling of more complex physical effects, such as non-linear effects.

The package `Basic` contains only one model level (model `Basic`), which is the simplest model for an AC electrical machine. Indeed, model `Basic` is defined by means of the torque constant and its behavior is ideal (losses and non-linear effects are neglected). Package `PermanentMagnet` includes the three-phase PMSM models and three model levels, such as `Standard`, `Saturation` and `Fault` models, are provided. Three-phase SRM models are organized in the package `Reluctance` and two model levels (`Standard` and `Saturation`) are implemented. Finally, the package `DirectCurrent` considers the DC machine models with independent excitation. This package consists of two modelling levels (`Standard` and `Saturation`). As previously underlined, the sub-package `Examples` is provided for each of the mentioned packages, in order to support the user in both familiarizing with the models and exploring their features.

It is worthy to remark that the contribution of this work is mainly focused on the package `PermanentMagnet` and all the considerations presented in the coming sections are made considering PMSM.

## 3 Models Interfaces

As mentioned earlier, every model belonging to a level is replaceable with one belonging to another level. This feature has been achieved by adopting common interfaces, which are defined in `Actuator.Electrical.Interfaces`. The adopted model interfaces can be classified according their function in:

- electrical interface;
- mechanical interface;
- thermal interface.

All quantities accessible at the mentioned interfaces are shown in physical units.

Considering AC electrical machines (packages `Basic`, `PermanentMagnet` and `Reluctance`), the electrical interfaces adopted are the `PositivePlug` and `NegativePlug`, since the stator winding is assumed as three-phase winding. In particular,

the AC machine models have been built up considering a wye-connected stator winding, with floating neutral point. The adopted electrical interfaces allow connecting the AC machine to a power converter or to a driven voltage source (`SignalVoltage`). In the case of the DC machine models, `PositivePin` and `NegativePin` have been used as electrical interfaces, for both armature and field windings (the DC machine excitation is independent).

All the machine models adopt `Flange` and `Support` as mechanical interfaces and they are included in the models by extending the `PartialOneFlangeAndSupport`. The developed electromagnetic torque and the load torque are applied on the machine `Flange`, while the reaction torque is available on the `Support`, in case the machine housing is not grounded.

The `PartialConditionalHeatPort` is used as thermal interface for all the machine models and housing temperature and heat flow are available at the heat port.

## 4 Model Basic

The model `Basic` is included in the package `Basic` and it is the simplest model for a generic three-phase AC machine. The machine behavior is described through the torque constant, which establishes the relationship between current and electromagnetic torque. Copper, iron and mechanical losses are not considered at this level, as well as, effects such as magnetic saturation, torque ripple and fault conditions are neglected.

Data records have been used to organize the model parameters and for the model `Basic` the required parameters are:

- Phase resistance ( $R$ );
- Magnetizing inductance ( $L$ );
- Pole pairs number ( $pp$ );
- Torque constant ( $k_t$ );
- Rotor's moment of inertia ( $J_r$ );
- Stator's moment of inertia ( $J_s$ ).

The stator's moment of inertia will be used by the model only when the machine housing is not grounded. Since plugs are used as electrical interfaces, the model `Basic` is fed using phase quantities ( $abc$ ) and Park's transformation (invariant amplitude transformation) [6] is adopted for implementing the model in the rotating reference frame ( $dq$ ), according to the following equations:

$$\begin{aligned} v_d &= R \cdot i_d + L \cdot \frac{d}{dt} i_d - \omega_e \cdot L \cdot i_q \\ v_q &= R \cdot i_q + L \cdot \frac{d}{dt} i_q + \omega_e \cdot L \cdot i_d + \frac{2 \cdot k_t}{3 \cdot pp} \cdot \omega_e \end{aligned} \quad (1)$$

where  $\omega_e$  is the electrical speed in  $rad/s$ . The electromagnetic torque ( $\tau_e$ ) developed by the model `Basic` is given by:

$$\tau_e = k_t \cdot i_q \quad (2)$$

The electrical equations are implemented in the *Modelica Text Layer*, while the mechanical model is realized in the *Modelica Diagram Layer*, using an object-oriented approach. In figure 2, the *Modelica Diagram Layer* of the model `Basic` is shown. Model `Basic`, as well as all the other machine models presented in the `Electrical Machines` library, is reversible. This means that the model `Basic` can work reversibly as motor or generator.

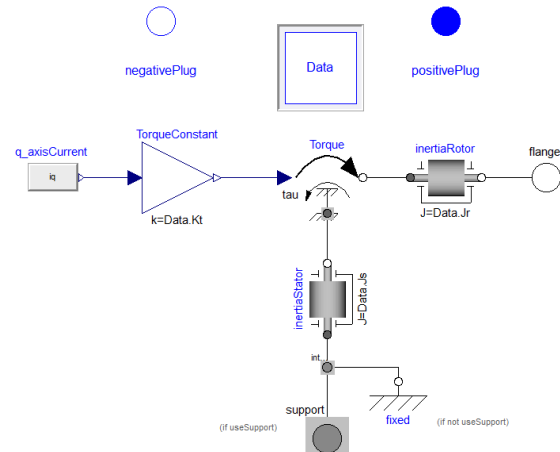


Figure 2: *Modelica Diagram Layer* of the model `Basic` included in the `Electrical Machines` library.

In order to verify the model `Basic` functionality, the simple EMA architecture, reported in figure 3, has been implemented and simulated in Dymola. The chosen EMA architecture consists of an inverter, which supplies a PMSM mechanically coupled to a 500  $rad/m$  transmission ratio roller screw, through a gearbox with transmission ratio equal to 5. The inverter is fed using a 270 V DC link and the PMSM parameters are: 14 pole pairs, 56.2  $g \cdot m^2$  rotor inertia, 0.05  $\Omega$  phase resistance, 2 mH magnetizing inductances and 2.1  $Nm/A$  torque constant. Control unit (*controller*) implements the conventional position control, hence the position is controlled by a cascaded structure, where the inner loop controls the  $q$ -axis current, the outer loop controls the speed and finally the outermost loop controls the position. During the simulation, the controlled sur-

face is moving 0.02  $m$  forward and backward, while an external load force of -80000  $N$  (equivalent to -32  $Nm$  load torque on the PMSM shaft) is acting on the flight surface. The simulation results are shown in figure 4. In particular the mechanical quantities (position, speed and torque) of the PMSM are reported and a good agreement between actual and reference position is highlighted.

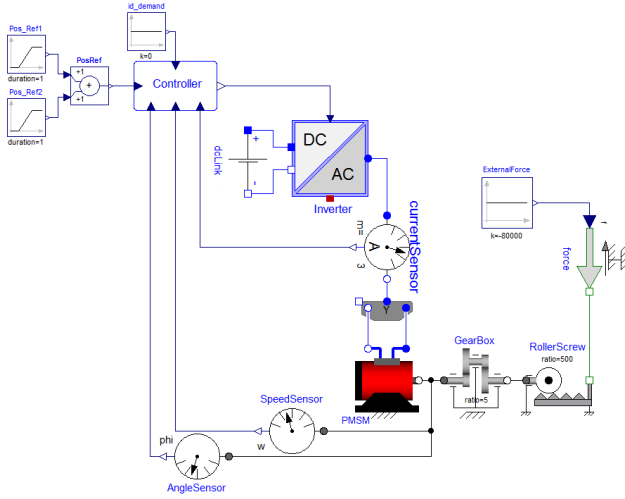


Figure 3: EMA architecture simulated in Dymola, using the model Basic.

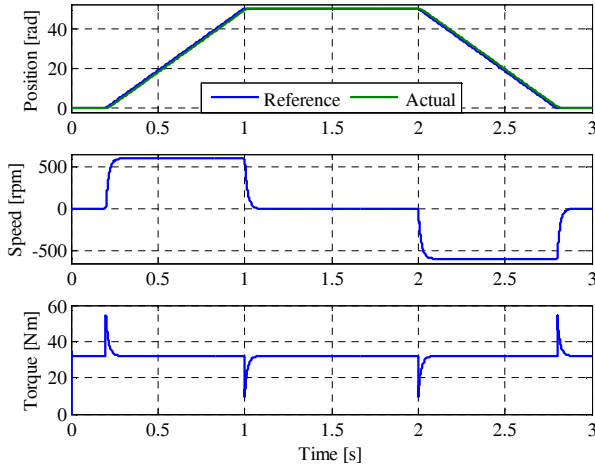


Figure 4: Simulation results obtained using the model Basic under position control: mechanical rotor position (top), mechanical rotor speed (middle) and developed torque (bottom).

## 5 Model Standard

The model Standard, included in the package PermanentMagnet, is based on a lumped parameters approach and it describes the behavior of PMSMs taking into account torque generation, losses and thermal model. Non-linear effects, such as magnetic saturation, torque ripple and fault conditions are neglecting at this level.

The model is developed in the rotating reference frame synchronous with the rotor ( $dq$ ) and Park's transformation (invariant amplitude transformation) is used to pass from the  $abc$  to the  $dq$  reference frame. Figure 5 reports the equivalent circuits in the rotating reference frame.

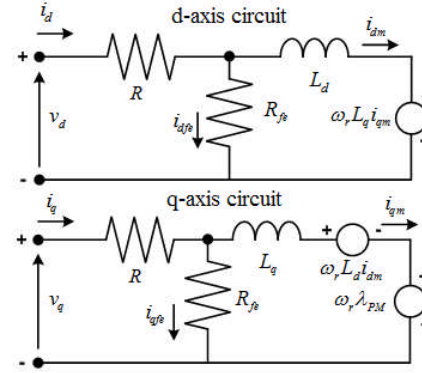


Figure 5:  $d$  – and  $q$  – axis equivalent circuits implemented within the model Standard.

These equivalent circuits take into account the iron losses using an equivalent iron losses resistance ( $R_{fe}$ ), placed in parallel with the series between the magnetizing branch and the electromotive force term [7]. Applying the Kirchhoff's voltage law (KVL) at the main loop of the circuits in figure 5, the following equations can be written:

$$v_d = R_t \cdot i_d + L_d \cdot \frac{d}{dt} i_{dm} - \omega_e \cdot L_q \cdot i_{qm} \quad (3)$$

$$v_q = R_t \cdot i_q + L_q \cdot \frac{d}{dt} i_{qm} + \omega_e \cdot L_d \cdot i_{dm} + \omega_e \cdot \lambda_{PM}$$

where  $R_t$  is the phase resistance (which is function of the winding temperature),  $\lambda_{PM}$  is the permanent magnet flux,  $L_d$  and  $L_q$  are the  $d$  – and  $q$  – axis magnetizing inductances respectively. These parameters ( $R$ ,  $L_d$ ,  $L_q$  and  $\lambda_{PM}$ ) are provided by the model user, through the model's mask. On the other hand, applying the Kirchhoff's current law (KCL) at the circuit nodes results:

$$\begin{aligned} i_d &= i_{dfe} + i_{dm} \\ i_q &= i_{qfe} + i_{qm} \end{aligned} \quad (4)$$

According (4), the total currents ( $i_d$  and  $i_q$ ) are split into iron losses currents ( $i_{dfe}$  and  $i_{qfe}$ ), given by (5), and magnetizing currents ( $i_{dm}$  and  $i_{qm}$ ). The formers flow through the equivalent iron losses resistance, while the latter are responsible of the torque generation.

$$\begin{aligned} i_{dfe} &= \left( L_d \cdot \frac{d}{dt} i_{dm} - \omega_e \cdot L_q \cdot i_{qm} \right) / R_{fe} \\ i_{qfe} &= \left( L_q \cdot \frac{d}{dt} i_{qm} + \omega_e \cdot L_d \cdot i_{dm} + \omega_e \cdot \lambda_{PM} \right) / R_{fe} \end{aligned} \quad (5)$$

In order to complete the PMSM model, the electromagnetic torque equation has to be explicit. In general, the torque developed by the PMSM is the sum of two terms: the field torque due to the permanent magnets and the reluctance torque due to the machine saliency. PMSM model Standard includes both these torque terms, hence the behavior of surface mounted (SMPM) and interior permanent magnet (IPM) machines can be simulated. The electromagnetic torque is given by (6):

$$\tau_e = \frac{3}{2} \cdot pp \cdot \left[ \lambda_{PM} \cdot i_{qm} + (L_d - L_q) \cdot i_{dm} \cdot i_{qm} \right] \quad (6)$$

where  $pp$  is the pole pairs number, which is provided by the model user.

Model Standard is a power balanced model and the losses taken into account are copper, iron and mechanical losses. Copper losses ( $P_{cu}$ ) are a function of the current and the winding resistance, according to the following equation:

$$P_{cu} = \frac{3}{2} \cdot R_t \cdot (i_d^2 + i_q^2) \quad (7)$$

The coefficient  $3/2$  arises from the adoption of the amplitude invariant Park's transformation. As previously said, the phase resistance  $R_t$  is function of winding temperature ( $\theta_w$ ), through the equation:

$$R_t = R \cdot \left[ 1 + \alpha \cdot (\theta_w - \theta_{ref}) \right] \quad (8)$$

where  $R$  is the phase resistance at the reference temperature ( $\theta_{ref}$ ) and  $\alpha$  is linear temperature coefficient, which depends on the winding material (i.e. copper or aluminum).

Iron losses ( $P_{fe}$ ), both hysteretic and eddy current, are a function of voltage and frequency as well as the material composition. These losses are calculated as equivalent copper losses:

$$P_{fe} = \frac{3}{2} \cdot R_{fe} \cdot (i_{dfe}^2 + i_{qfe}^2) \quad (9)$$

Equivalent iron losses resistance is obtained considering the specific iron losses ( $C_p$ ) and the stator mass ( $m_s$ ):

$$R_{fe} = \frac{(2 \cdot \pi \cdot 50)^2 + \lambda_{PM}^2}{C_p \cdot m_s} \quad (10)$$

Both these parameters ( $C_p$  and  $m_s$ ) are provided by the model user, using the model's mask. Specific iron losses gives the iron losses in 1 Kg mass of a given magnetic material, when it is subject to a flux density of 1 Wb/m<sup>2</sup> at 50 Hz.

Finally, mechanical losses ( $P_{mec}$ ), due to friction and windage, are functions of rotor speed and they are included in the model using the equivalent viscous friction torque ( $\tau_f$ ):

$$\tau_f = B \cdot \frac{\omega_e}{pp} \quad (11)$$

$$P_{mec} = \tau_f \cdot \frac{\omega_e}{pp} \quad (12)$$

where  $B$  is the friction coefficient. The stray losses are neglected in the model Standard, due to their small amount compared to the other losses.

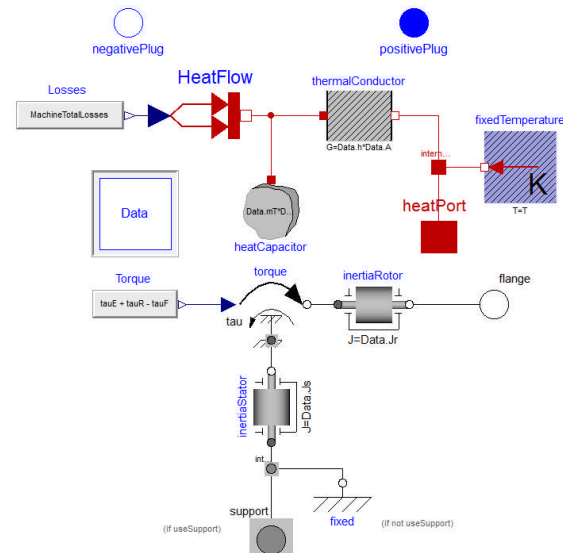


Figure 6: Modelica Diagram Layer of the model Standard included in the Electrical Machines library.

Total losses are applied in input to the thermal model adopting the component PrescribedHeatFlow. The implemented thermal model considers the PMSM as a whole block and it is composed by a heat capacitor and a thermal conductor. The heat capacity ( $C$ ) is defined as:

$$C = c_p \cdot m_T \quad (13)$$

where  $m_T$  is the total motor mass and  $c_p$  is the equivalent specific heat capacity, which depends by the motor materials (mainly copper and iron). However, the convective thermal conductance ( $G$ ) between windings and housing is given by:

$$G = h \cdot A \quad (14)$$

where  $A$  is the convection area and  $h$  is the heat transfer coefficient, which depends upon the cooling system adopted. Both these parameters, together with  $m_T$  and  $c_p$  are provided by the model user. The winding and housing temperatures are respectively available on `port_a` and `port_b` of the thermal conductor. Finally, the heat flow is available at the heat port of the thermal model (`heatPort`). Equations (3)-(14) are implemented in the *Modelica Text Layer*, while the mechanical and thermal models are included in the *Modelica Diagram Layer*, as shown in figure 6.

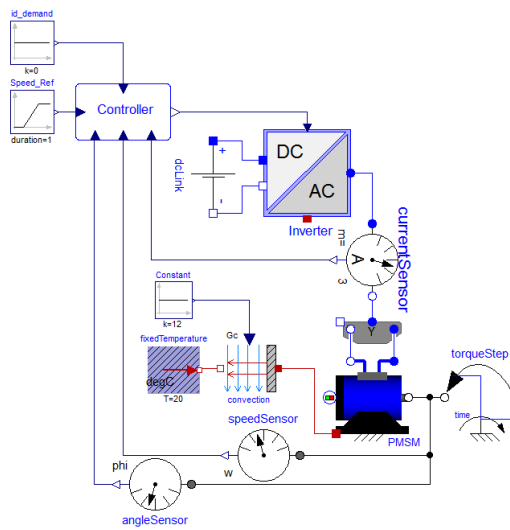


Figure 7: Speed control scheme simulated in Dymola, using the model Standard.

Model Standard has been simulated under speed control using the block diagram reported in figure 7, with the purpose of highlighting the PMSM losses and the thermal behavior. The PMSM parameters are:

- pole pairs number 14;
- rotor's moment of inertia  $56.2 \text{ g} \cdot \text{m}^2$ ;
- phase resistance  $0.05 \text{ } \Omega$ ;
- $d$  – and  $q$  – axis inductance  $2 \text{ mH}$ ;
- permanent magnet flux  $0.1 \text{ Wb}$ ;
- specific iron losses  $1.1 \text{ W/kg}$  (M300);
- stator mass  $3 \text{ kg}$ ;
- friction coefficient  $0.0002 \text{ Nm} \cdot \text{s}$ ;
- motor mass  $5 \text{ kg}$ ;
- specific heat capacity  $424 \text{ J/(kg} \cdot \text{K)}$ ;
- convection area  $0.75 \text{ m}^2$ ;
- heat transfer coefficient  $12 \text{ W/(K} \cdot \text{m}^2)$ .

During the simulation, the mechanical speed reference signal is a ramp from  $0 \text{ rpm}$  (at  $0.01 \text{ s}$ ) to  $1500 \text{ rpm}$  (at  $1.01 \text{ s}$ ), while the  $d$  – axis current

reference signal is kept equal to  $0 \text{ A}$ . Once the speed reaches the steady state value ( $1500 \text{ rpm}$ ), a load torque step of  $-25 \text{ Nm}$  is applied at  $1.5 \text{ s}$ . Figure 8 shows the PMSM losses (copper, iron and mechanical losses) together with the electrical and mechanical powers. In steady state, the machine efficiency is equal to  $93.45\%$ .

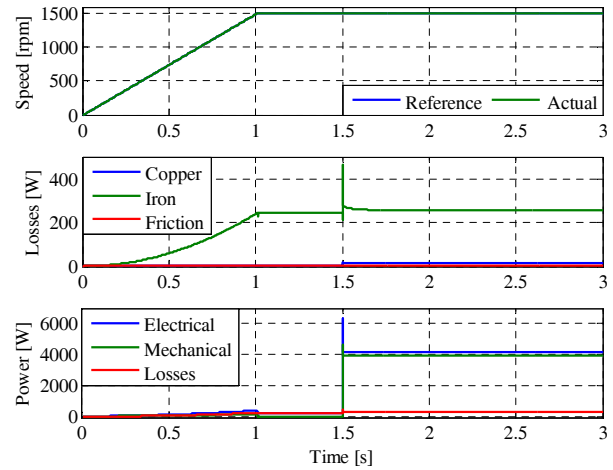


Figure 8: Simulation results obtained using the model Standard under speed control: mechanical rotor speed (top), PMSM losses (middle) and PMSM power (bottom).

The trends of the windings and housing temperatures, over a working period of  $1 \text{ hr}$ , are reported in figure 9.

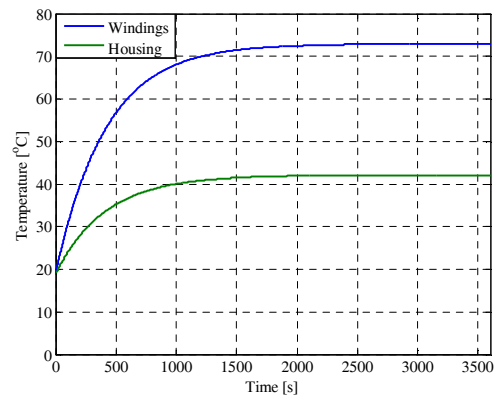


Figure 9: Simulation results obtained using the model Standard: winding and housing temperatures.

## 6 Model Saturation

The model Saturation, included in the package PermanentMagnet, has been obtained as an extension of the model Standard. Therefore it takes into account all the physical effects already presented in the model Standard section, plus the magnetic saturation and the torque ripple. As with the model Standard, the model Saturation is also developed in the rotating reference frame ( $dq$ ) and Park's transformation (amplitude invariant trans-

formation) is used to pass from the  $abc$  to the  $dq$  reference frame.

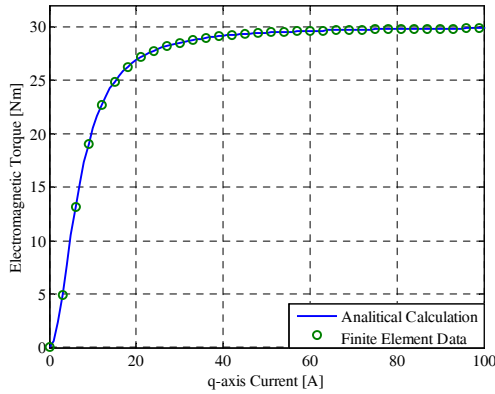


Figure 10: Field torque component against  $q$ -axis current: analytical curve (blue line) and finite element results (green dots).

In ferromagnetic materials, the relationship between the field strength and flux density is not a straight line (i.e. constant permeability), but it is defined through the magnetization curve. Since ferromagnetic materials reveal a variable permeability according to the field strength, the magnetization curve has a knee point. Past this knee point, further increases in the field strength do not result in proportional increases in flux density. The PMSM performance is affected by the magnetic saturation; in particular, it influences both the permanent magnet flux and the inductances. Since the permanent magnet flux and the inductances are a function of the operating point, the developed torque (both field and reluctance terms) is not proportional to the current anymore.

In the model Saturation for PMSM, the magnetic saturation is taken into account considering only its influence on field torque term (i.e. permanent magnet flux), while the reluctance torque term (i.e. magnetic saliency) is not affected by the magnetic saturation. In the presence of magnetic saturation, the curve of the field torque term against the  $q$ -axis current (torque-current curve) has a trend, like shown in figure 10. The torque-current curve can be included in the model Saturation using two approaches:

- the look-up table;
- three given points, which identify the torque-current curve.

In the former case, the data stored inside the look-up table may be obtained by finite element simulations. In the latter case, the model user provides the following data:

- $q$ -axis current at the knee of the torque-current curve ( $I_{qm\text{knee}}$ );

- field torque at the knee of the torque-current curve ( $T_{\text{knee}}$ );
  - field torque at the saturated region of the torque-current curve ( $T_{\text{sat}}$ );
- and the developed torque is analytically calculated using (15).

$$\tau_e = \frac{\sqrt{c_1 \cdot i_{qm}^4 + i_{qm}^2}}{c_2 + i_{qm}^2} \quad (15)$$

where  $i_{qm}$  is the actual magnetizing current along the  $q$ -axis, while  $c_1$  and  $c_2$  are the torque function coefficients. These coefficients are expressed as shown below.

$$c_1 = T_{\text{sat}} \quad (16)$$

$$c_2 = \sqrt{\frac{I_{qm\text{knee}}^4 \cdot T_{\text{sat}}^2 + I_{qm\text{knee}}^2 - I_{qm\text{knee}}^2 \cdot T_{\text{knee}}}{T_{\text{knee}}}} \quad (17)$$

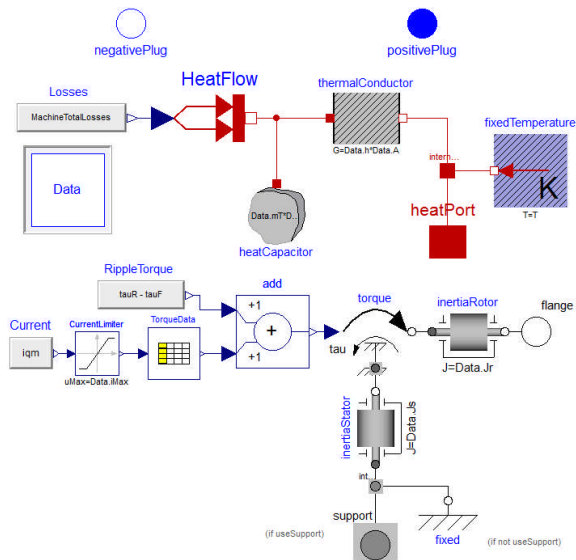


Figure 11: Modelica Diagram Layer of the model Saturation included in the Electrical Machines library.

In addition to the magnetic saturation, the model Saturation, also implements the torque ripple, which is function of the electrical rotor position ( $\vartheta_e$ ). In particular, the model Saturation allows to superimpose a sinusoidal torque ripple ( $\tau_{\text{ripple}}$ ) on the developed electromagnetic torque ( $\tau_e$ ). The torque ripple waveform is defined according (18).

$$\tau_{\text{ripple}} = A_{\text{ripple}} \cdot \sin\left(\frac{f_{\text{ripple}}}{f_n} \cdot \vartheta_e\right) \quad (18)$$

where  $A_{\text{ripple}}$  is the ripple amplitude,  $f_{\text{ripple}}$  is the ripple frequency and  $f_n$  is the PMSM rated frequency. These parameters are provided by the model user by

means of a dedicated mask.  $A_{ripple}$  and  $f_{ripple}$  are function of [8]:

- stator winding (concentrated or distributed, single or double layer);
- motor geometry (slot opening, tooth shape, etc...);
- air-gap flux density quality (harmonic content);
- load conditions.

Equations (15)-(18) are implemented in the *Modelica Text Layer*, while the look-up table is placed in the *Modelica Diagram Layer*, as shown in figure 11.

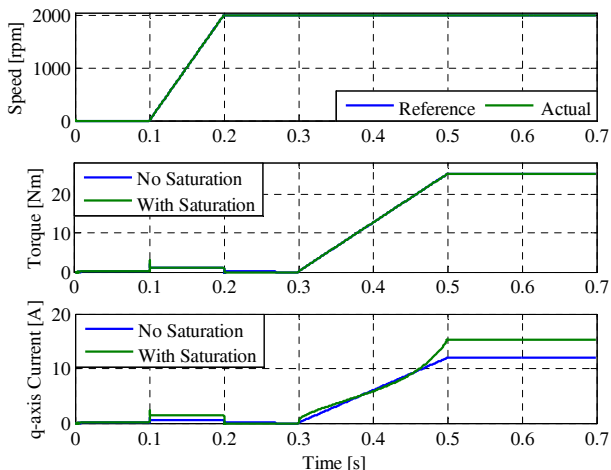


Figure 12: Simulation results obtained using the model Saturation under speed control: mechanical rotor speed (top), developed torque (middle) and  $q$  – axis current (bottom).

In order to verify the magnetic saturation influence on the PMSM performance, model Saturation has been simulated under speed control within Dymola. PMSM parameters are the same adopted for the simulation test shown in the previous section, while the magnetic saturation parameters are:

- $q$  – axis current at the knee equal to 5 A ;
- field torque at the knee equal to 10.5 Nm ;
- field torque in the saturated region equal to 30 Nm .

These parameters define the torque-current curve reported in figure 10 (blue line). During the simulation, the mechanical speed reference signal is a ramp from 0 rpm (at 0.1 s) to 2000 rpm (at 0.2 s), while the  $d$  – axis current reference signal is kept equal to 0 A. Once the speed reaches the demanded steady state value (2000 rpm), a load torque ramp from 0 Nm (at 0.3 s) to -25 Nm (at 0.5 s) is applied, in order to emphasize the magnetic saturation effect. Figure 12 shows the simulation results. In particular, the comparison between the  $q$  – axis currents flowing into the PMSM, with (green line) and

without magnetic saturation (blue line), is highlighted. When the magnetic saturation is considered, in order to develop the same electromagnetic torque (as in the absence of magnetic saturation), a higher  $q$  – axis current is required. Even during the acceleration phase, the  $q$  – axis current draw is higher when the magnetic saturation is taken into account. This is due to the analytical implementation of the torque-current curve (15). Indeed, torque-current curve is not a straight line in the linear region (see figure 10).

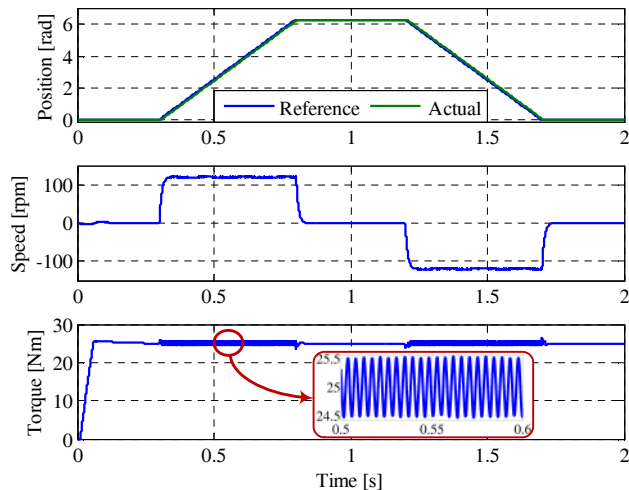


Figure 13: Simulation results obtained using the model Saturation under position control: mechanical rotor position (top), mechanical rotor speed (middle) and developed torque (bottom).

Figure 13 reports the simulation results obtained when the PMSM is position controlled and the torque ripple is considered. Torque ripple parameters have been set equal to 0.5 Nm amplitude (2% ripple) and 11200 Hz frequency. During the simulation, a -25 Nm load torque is applied. The enlargement in figure 13 (bottom plot) highlights the superimposition of the torque ripple on the developed torque. Torque oscillations, due to the ripple, have repercussions on both mechanical speed (middle plot) and  $q$  – axis current.

## 7 Models Fault

Models including fault conditions have been implemented only for PMSM (package Permanent-Magnet), since the presence of permanent magnets is the source of concern in the event of fault conditions. Indeed, fault condition is supplied by the back electromotive force due to the permanent magnets [9]. PMSM models including fault conditions have been implemented in the phase reference frame ( $abc$ ), since asymmetric faults are considered as well. Winding short-circuits, winding open-circuits and permanent magnet demagnetization are the three



fault conditions considered during the failure modeling. For each of the mentioned fault conditions a specific model has been developed, in order to keep low the model complexity. The realized models are listed below:

- model `ShortCircuit`, which implements single- and three-phase short-circuits;
- model `OpenCircuit`, which considers single- and three-phase open-circuits;
- model `Demagnetization`, which takes into account the permanent magnet demagnetization.

As suggested in [10], the winding faults have been implemented by means of ideal electrical switches. Such switches are driven by Boolean variables in order to open or close the phase windings and inject the fault conditions. Figures 14 and 15 show the *Modelica Diagram Layer* of model `ShortCircuit` and model `OpenCircuit` respectively.

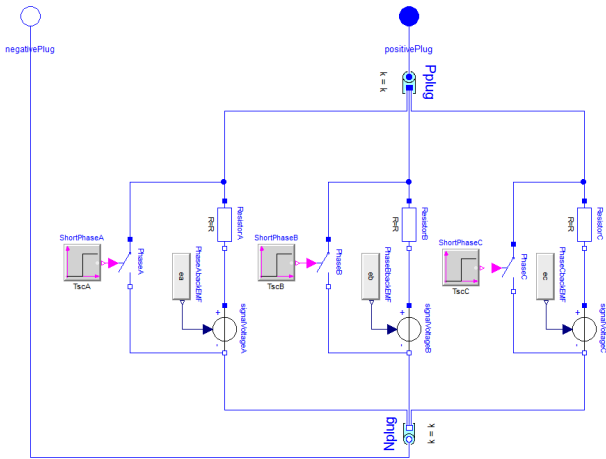


Figure 14: *Modelica Diagram Layer* of the model `ShortCircuit` included in the Electrical Machines library.

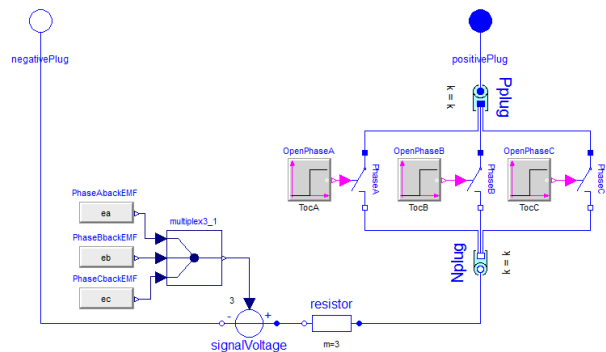


Figure 15: *Modelica Diagram Layer* of the model `OpenCircuit` included in the Electrical Machines library.

Demagnetization is caused by increasing of temperature inside the permanent magnet and/or high current values. In the model `Demagnetization`, the fault condition is considered as consequence of the fault

(i.e. permanent magnet flux reduction) neglecting the causes that led to the fault.

Figure 16 shows the Dymola block diagram used for testing the models including fault conditions. In particular, PMSM is current controlled (with  $i_q = 10 A$  and  $i_d = 0 A$ ) and its rotor is speed driven at a constant  $500 rpm$ . For the sake of brevity, only the simulation results regarding model `ShortCircuit` and model `OpenCircuit` are reported here. Moreover, the PMSM parameters are the same as those adopted in the model `Standard`, within section 5. Using model `ShortCircuit`, a three-phase short-circuit (symmetric fault) has been injected at  $0.3 s$ . By this time, the mechanical speed and currents have reached the steady state values. The corresponding simulation results are reported in figure 17. From the top plot, it is evident that the rotor speed is kept constant even after the fault injection. The middle plot shows the braking torque ( $T_{br}$ ) and its the steady state value is equal to  $-3.57 Nm$ . The same value can be obtained applying the formula reported in [11] and written below:

$$T_{br} = -\frac{3}{2} \cdot pp \cdot R \cdot \lambda_{PM}^2 \cdot \omega_e \cdot \frac{R^2 + \omega_e^2 \cdot L_q^2}{(R^2 + \omega_e^2 \cdot L_d \cdot L_q)^2} \quad (19)$$

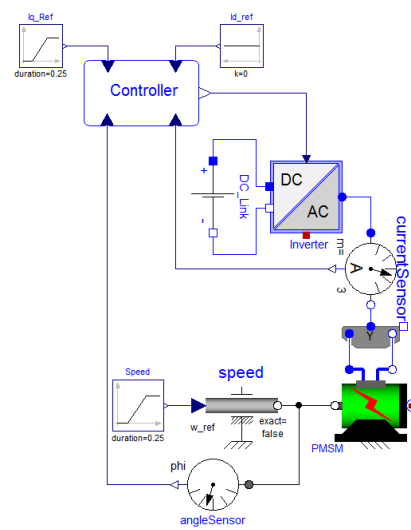


Figure 16: Block diagram implemented in Dymola, for testing both model `ShortCircuit` and `OpenCircuit`.

Since the three-phase short-circuit is a symmetric fault, the current components in the  $dq$  reference frame can be defined (see bottom plot). The steady state values of the  $dq$  current components are  $i_{qsh} = -1.7 A$  and  $i_{dsh} = -49.9 A$ . These values are in line with those obtained applying the analytical formulas reported in [11].

$$I_{dsh} = -\frac{\omega_e^2 \cdot \lambda_{PM} \cdot L_q}{R^2 + \omega_e^2 \cdot L_d \cdot L_q} \quad (20)$$

$$I_{qsh} = -\frac{\omega_e \cdot \lambda_{PM} \cdot R}{R^2 + \omega_e^2 \cdot L_d \cdot L_q} \quad (21)$$

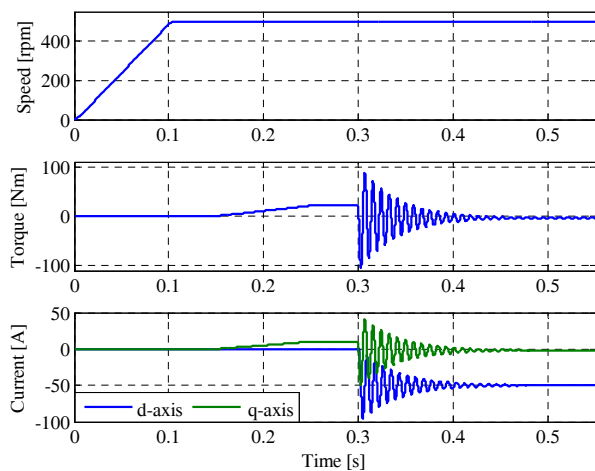


Figure 17: Simulation results obtained using the model `ShortCircuit`, in case of three-phase short-circuit: mechanical rotor speed (top), developed torque (middle) and  $dq$  currents (bottom).

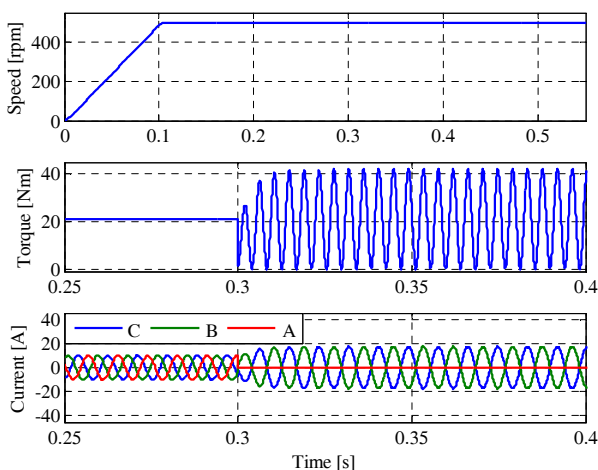


Figure 18: Simulation results obtained using the model `OpenCircuit` in case of single-phase open-circuit: mechanical rotor speed (top), developed torque (middle) and  $abc$  currents (bottom).

In figure 18, the simulation results obtained using model `OpenCircuit` are reported. In this case, a single-phase open-circuit (asymmetric fault) is injected into phase  $a$  at  $0.3$  s. After the fault injection, the developed torque (middle plot) becomes oscillating, since the PMSM currents are unbalanced. The pulsating torque component is shown alongside the demanded torque under healthy conditions ( $21$  Nm). Its frequency is twice the operating frequency of the PMSM. Since the stator winding is assumed wye-connected, with a floating neutral point, and the current in phase  $a$  is null, due to the

phase opening, the phase currents in phases  $b$  and  $c$  must be shifted by 180 degrees. The described situation can be observed in the bottom plot of figure 18.

## 8 Conclusions

An electrical machines library developed using Modelica has been presented in this paper. Particular attention has been paid to the PMSM models, since PMSMs are becoming widespread in aerospace applications, due to their inherent features. A multi-level approach has been adopted, in order to keep the model complexity low and reduce the computation time. The interchangeability among modelling levels has been ensured by using common interfaces. All the models are power balanced and can work as either a motor or generator. The main physical effects taken into account are losses, thermal behavior, magnetic saturation, torque ripple and fault conditions. Physical effects included in the models have been discussed and their implementation has been detailed. The effectiveness of the various modelling levels has been proven through simulation results obtained in several operating conditions.

## References

- [1] A. Boglietti; A. Cavagnino, A. Tenconi and S. Vaschetto, "The safety critical electric machines and drives in the MEA: A survey", in *IEEE Ind. Elec.*, Nov. 2009.
- [2] S.V. Bozhko, T. Wu, Y. Tao and G.M. Asher "MEA electrical power system accelerated functional modelling", in *Power Electr. and Motion Control*, Sep. 2010.
- [3] P. Fritzson, "Principles of Object-Oriented Modeling and Simulation with Modelica 2.1", IEEE Press, 2004.
- [4] J.F. Gieras "PM Motors Technology: Design and Applications", 3<sup>rd</sup> edition, Taylor and Francis, 2010.
- [5] C. Gerada and K.J. Bradley, "Integrated PM Machine Design for an Aircraft EMA", *IEEE Trans. on Ind. Elect.*, vol. 55, no. 9, pp.3300-3306, Sept. 2008.
- [6] R.H. Park, "Two-reaction theory of synchronous machines generalized method of analysis-part I", *Trans. of the American Institute of Electrical Engineers*, vol. 48, no. 3, pp. 716-727, July 1929.
- [7] N. Urasaki, T. Senjyu and K. Uezato "A Novel Calculation Method for Iron Loss Resistance Suitable in Modeling PMSMs" *IEEE Trans. on Energy Conversion*, vol. 18, no. 1, pp. 41-77, March 2003.
- [8] G. Ferretti, G. Magnani and P. Rocco, "Simulating permanent magnet brushless motors in Dymola", 2<sup>nd</sup> Modelica Conference, pp. 109-115, March 2002.
- [9] S. Khwan-on, L. de Lillo, L. Empringham, P. Wheeler, C. Gerada, N.M. Othman, O. Jasim, and J. Clare, "FT power converter topologies for PMSM drives in aerospace applications", 13<sup>th</sup> EPE, pp. 1-9, 8-10 Sept. 2009.
- [10] D. Winkler and C. Gühmann, "Modelling of Electrical Faults in IMs Using Modelica", 48<sup>th</sup> Scandinavian Conference on Simulation and Modeling, 2007.
- [11] N. Bianchi, M.D. Pr e and S. Bolognani, "Design of a fault-tolerant IPM motor for electric power steering", *IEEE Trans. on Vehicular Technology*, vol. 55, no. 4, pp. 1102-1111, July 2006.

# Modelica for large scale aircraft electrical network V&V

Martin R. Kuhn   Yang Ji  
German Aerospace Center (DLR), Institute of System Dynamics and Control  
Münchnerstr. 20, 82234 Weßling  
martin.kuhn@dlr.de   yang.ji@dlr.de

## Abstract

More electrically powered aircraft reveals some significant advantages such as weight decrease, reduced maintenance requirements and increased reliability and passenger comfort. However, the development of the future more-electric aircraft (MEA) systems is a very challenging task. Its complexity may be handled by a model supported design approach for the total aircraft design process. A key factor of applying model based design is dedicated modeling and simulation techniques for all design phases. The highest complexity can be seen in the systems validation and verification phase where the aircraft system is integrated from the supplier's models.

While the capability to conveniently model complex physical systems with Modelica is generally accepted, the capability to perform large scale model integration and analysis as part of a validation and verification process remained unproven. In this paper we give evidence of Modelica/Dymola to be suitable for the virtual testing of complex energy systems in the future MEA design process. We demonstrate the modeling and the simulation results of component stand-alone tests as well as the tests of an integrated aircraft power network.

*Keywords:* V&V; electrical network; simulation; aircraft

## 1 Introduction

The model based design approach is a key factor for more efficient aircraft design with its growing demand to optimize the complex physical systems containing mechanical, electrical, hydraulic, thermal, control, electric power or process-oriented sub components [15]. Especially the more electric aircraft concept relies on incorporating high quality system models in the complete aircraft design process [1]. The process itself briefly can be divided into 4 major phases: con-

cept phase, system specification phase, system development phase and system verification phase [5].

- **Concept phase:** During concept phase a two-fold iterative optimization is performed on aircraft manufacturer side. This includes aircraft concept and global energy system architecture optimization.
- **System specification phase:** In this phase, a frozen energy system concept is provided by the aircraft manufacturer. Additionally, more detailed aircraft data about structure, cabin, light physics, engine and electrical power generation are available. The selected system suppliers conduct full concept definition where all the requirements and risks are understood. The aircraft manufacturer's requirements is transformed to the level of equipment suppliers by the system suppliers. Stability studies and failure analysis of aircraft electrical network are typical activities during the system specification phase.
- **System development phase:** In this phase preliminary and detailed design of equipment takes place. Verification and validation for artifacts are done, which are produced during this phase.
- **System verification phase:** The objective in the system verification phase is to demonstrate the maturity of the systems in a realistic integration and verification of more-electric aircraft systems, capable of covering all phases of the development process. The virtual integration platform for energy systems allows addressing integration issues prior to their physical integration on the test rigs and also extend test coverage. Power quality investigation of the integrated network will be of the interest in this phase. The typical tasks of the aircraft manufacturer in this phase are: monitor supplier system development by verification of system performance and functions, integrate sys-

tems in physical and functional aircraft, verify integrated systems and validate simulation models versus test results.

The model types and level of detail change for every phase. For example an advanced concept phase tool like ENADOT [14], applicable for optimal architecture design of the electrical energy system, in general does not demand more detailed electrical circuits than resistive elements. In contrast, the aircraft electrical network validation and verification process strongly relies on software for detailed and numerical complex modeling, simulation and analysis of network components and systems. Substantial efforts were made to reach platform independence and link simulation tools each with special strengths and dedicated for specific domains. Especially the FMI standard was a major step forward and was verified to improve an aircraft systems design process. Nevertheless, for the sake of performance and transparency, industrial processes often rely on a single common tool.

The software used in an aircraft project for the systems integration validation and verification (V&V) process is defined by the airframer for all model suppliers and contributors. While Modelica has found attraction in the automotive sector, it is not the standard for detailed simulation in aeronautic industry yet. Inspired by the success in the prior design phases, a study was performed in the context of the CleanSky project [4] to evaluate the potential and performance of Modelica and the commercial tool Dymola for electrical V&V. In this paper we give an overview of the necessities of the infrastructure which had to be developed. Necessary tools are addressed and lessons learned from the study are documented. It is the aim of this paper to rise awareness of the needs to conduct V&V studies. This paper quotes parts of [7, 6] with special focus on the Modelica community.

In chapter 2 the general procedure for model V&V is presented. The following paragraph gives an overview of the models and library structure. Some results and lessons learned from simulation are documented in the “simulation” chapter. It is concluded by an overview of the methods and tools developed for the study.

## 2 Procedure of virtual testing

In this chapter we want to reveal the general procedure of the V&V process. Details were published in [7].

Today, virtual testing of the integrated aircraft energy system is becoming an indispensable task in the

system verification design phase. The virtual testing procedure enables integration of the system by software before the real physical integration on the test rigs and extends test coverage. The behavior of the integrated system is estimated to be representative since component models are verified by in-house hardware tests at the suppliers. Each component is delivered as detailed (behavioural) and abstracted (functional) model.

The virtual testing process can be briefly divided into two steps. First, each subsystem or component model shall be tested for correct operation by so-called component standalone tests. A standalone test usually consists of a bunch of single tests such as power connection, power disconnection, power consumption at steady state, current harmonic analysis and so on, for one component. Standalone tests are required for both functional and behavioral models.

Once standalone tests for all components and subsystems are successful finished, in the next step simulations and tests of the total integrated system model are performed. Finally, specific analysis and post-treatment tasks can be performed based on the simulation results of the integrated models.

## 3 Modeling

To illustrate the type of system under investigation, a typical MEA energy system is depicted in figure 1. The system is powered by the variable frequency generator which controls the bus voltage by the generator control unit (GCU). The AC voltage is rectified by an auto transformer rectifier unit (ATRU) which feeds the environmental control system (ECS), DC loads and the direct current charging unit (DCCU) connecting low voltage DC loads. Low level AC voltage loads are supplied by an auto transformer unit (ATU).

### 3.1 Functional/behavioural model

As written before, different modeling levels apply for different test scenarios. Today, aircraft industry utilizes a three multi-level approach for the design of the aircraft system [10].<sup>1</sup>

The models are split into three types:

<sup>1</sup>To improve the international common understanding of modeling levels and modeling needs, SAE Aerospace organisation will publish a document titled “AIRCRAFT ELECTRICAL POWER SYSTEMS. MODELLING AND SIMULATION. DEFINITIONS.” in the near future

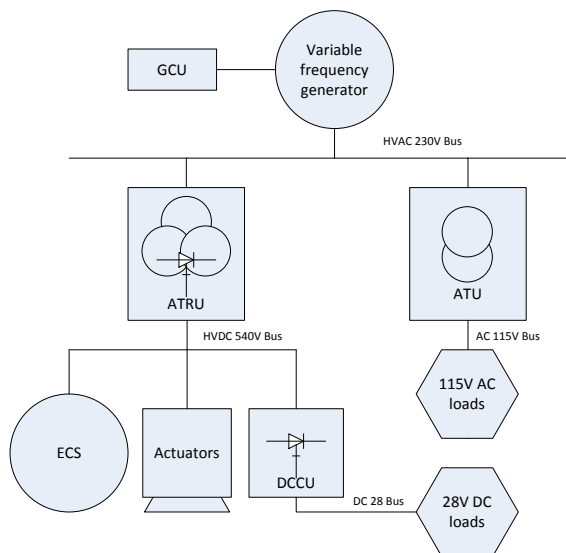


Figure 1: Electric network architectures for More Electric Aircraft

- Architectural models consist of algebraic equations and are used for steady-state power consumption calculations.
- Functional models reflect the low frequency behavior of the original system till around one third of the base grid frequency excluding switching ripples. Functional models are derived from behavioral models by state space time averaging of high frequency periodical switching waveforms. Typical applications are stability studies [12, 11] and control design. For the AC network in research often a dq equivalent network representation [9] is chosen. For an industrial project this might be further restricted to an equivalent one phase DC system. While this simplifies the system essentially by neglecting the AC phase information, no calculation of reactive power is possible .
- The behavioral models reflect both low and high frequency dynamics including switching effects. Behavioral models are based on equations derived from the subsystem structure and electrical circuit. The behavior at the terminals should be equivalent to the real hardware up to frequencies in the hundred kilohertz. Applications include power quality simulation and analysis of transient effects [8, 16].

The table 1 presents an overview of model requirements in all aircraft design phases.

For the system verification phase, functional models

are used for long term studies of the integrated system. Behavioral models are mainly used for detailed investigation of transients as power on phases.

### 3.2 Electrical components

The library of components and systems developed for the project can be seen in figure 2. The structure follows the needs of the V&V procedure: All models are implemented as a “behavioural” and “functional” representation.

All models have to be verified by associated standalone tests. The Modelica language concept showed to be beneficial in organizing integrated libraries with both, models and scripts. For example, in figure 2 the test routines for the 230VAC/115VAC auto-transformer rectifier model are emphasized. Scripts can contain procedures for parameter setting, simulation commands and post processing and documentation features. The newly developed scripts are addressed in chapter 5.3.

In the project it was confirmed, Modelica language and the Modelica standard library are capable of modeling all subsystems sufficiently. While the Modelica electrical library is known to be of limited size compared to design environments specialized for electrics, it was found out most components can be modeled by generic objects (e.g. rectifier unit) or they are very specific and need to be written textually by equations anyway (e.g. the generator). The only type of model which was missing showed to be a detailed magnetic hysteresis model. Magnetic hysteresis is of special importance for electrical power systems since the initial magnetizing effect of electrical transformers at power connection can lead to short-time excessive currents flowing into the transformer. This effect is called “inrush current” and investigated in [6]. The Modelica magnetic hysteresis model [17] was developed from JTI resources and will be part of the Modelica standard library in the future.

As an example for the library the ATU model is shown in figure 3 which is one of the critical magnetic elements. The component tests include harmonic current test, inrush current test, power connection and power disconnection test. The harmonic current analysis aims to determine disturbances due to the equipment on different frequency levels. Fast Fourier transformation (FFT) is performed after simulation of the ATU model reaches steady state condition (figure 4).

Design phase	Typical task	Required model
Concept	Architecture optimization	Level 1
System specification	Stability studies	Level 2/3
System development	Control design	Level 2/3
System verification	Virtual testing	Level 2/3

Table 1: Model requirements in different aircraft design phase

### 3.3 Integrated aircraft power system

The stand alone test are essential prerequisites to debug the single components for stable simulation before the integration. After successful stand-alone tests for all components, various scenarios for testing the complete electric power network can be performed. To demonstrate the capability of Modelica/Dymola to deal with large scale power systems, the proposed electric power network depicted in figure 5 has been simulated in Dymola at both behavioral and functional levels. The behavioral model is reduced by Dymola to a simulation model with 69 continuous time states. The linear system to be solved reduces to one equation system of order 18. The initialization system was reduced to five independent nonlinear equation systems where the largest was of order 33. While the numbers by itself seem not to be very impressive, complexity comes from the switching system.

In the demonstrated electric power network, the ATRU is connect to grid at 0.0025 second. After the pre-charging ATRU with 25e-3 second, the DC output of the ATRU is connected with the HVDC network. The PMSM which has a 20e-3 second pre-charging time is connected with the HVDC network at 0.055 second. After the power inverter in the PMSM is activated, a constant speed command is given for the PMSM under a constant load. AC currents and voltages of the VFG is recorded in the figure 6 for behavioral model and in figure 7 for the functional model. In the simulation results, it is clear to see the inrush currents at the moment of switching on ATRU and DC ripple at the ATRU output. These values are very important indicators for the stability study for the electric power network in MEA. [12]

## 4 Simulation and lessons learned

### 4.1 General

As benchmark of the study, the components and especially the large aircraft power systems had to be simulated to demonstrate the performance and robustness of Dymola's numerical solver for such a usually very

stiff power system and suffering from huge amount of event handling actions due to switching components.

In the study it could be demonstrated Modelica/Dymola is capable of simulating all component and integration tests. Compared to the traditional simulation platform for V&V tests, numerical speed showed to be excellent for the smaller component tests and competitive for the large integration test and might be further improved. Also it was detected, simulation speed is overwhelmingly dependent on the model quality and the experience of the designer. Stable and fast operation of the components with non-specialized integration algorithms as DASSL was almost mandatory for the successful large system integration.

From the example in the previous chapter it was seen, the complexity of the initialization in many cases prevents simulation already before start. As a workaround the designer may test initialization with the steady state option.

### 4.2 Identification of modeling errors

A typical problem for large scale simulation is abortion, slow progress of simulation or chattering caused by modeling problems. For a limited number of modeling errors, the simulator's log usually gives important hints while they are not easy to interpret. For example, in the V&V study dynamic state selection was indicated by the simulator for a (working) component model. For the large scale test this inhibited numerical problems. This phenomenon and a workaround for the magnetic system is documented in [ifeec]. As a second example, for the generator model standalone test algebraic loops were indicated in the simulator's log. Algebraic loops can occur for feedback loops with feedthrough part, which can not be simplified by the symbolic routines. While algebraic loops are unusual for standard electric circuits they appear easily for saturated elements. In the V&V study the equation system conditioned by the algebraic loop accounted for numerical problems. While it does not necessarily help, there is the option to cut the loop by insertion of a first order delaying element.

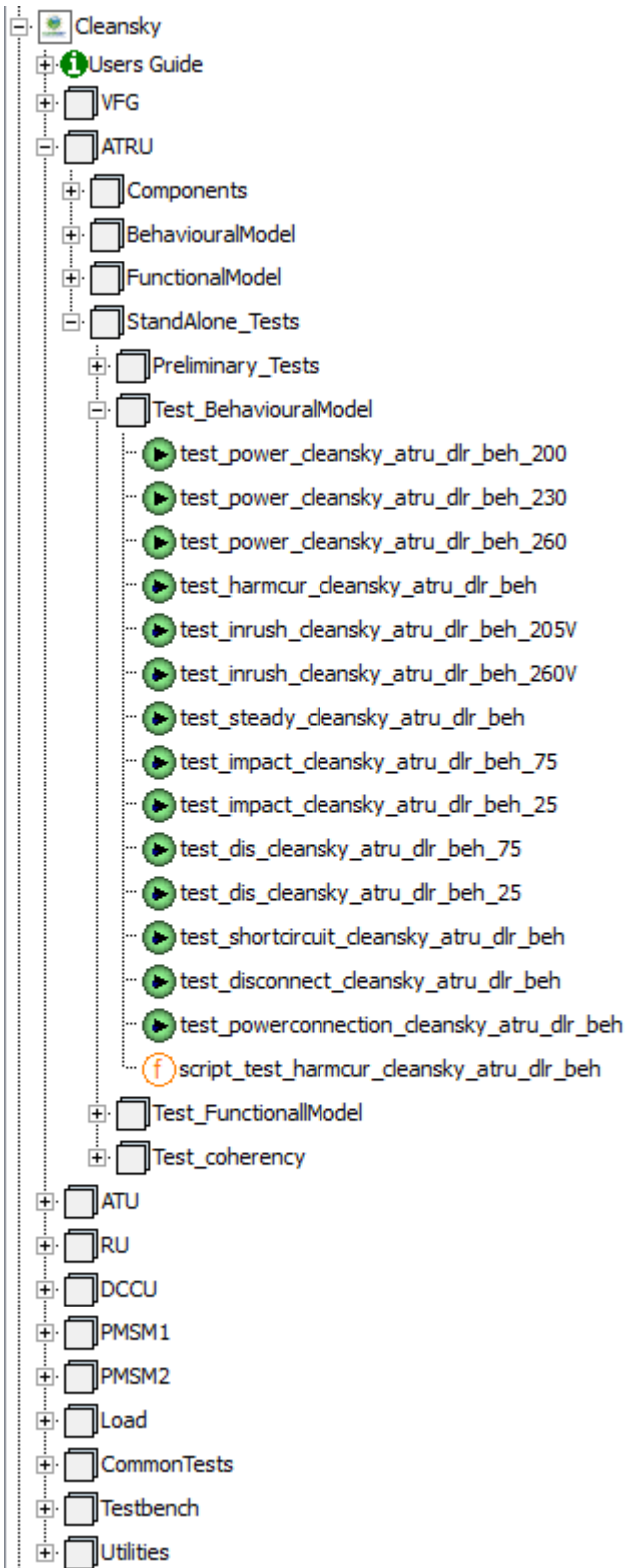


Figure 2: Library of the electrical system

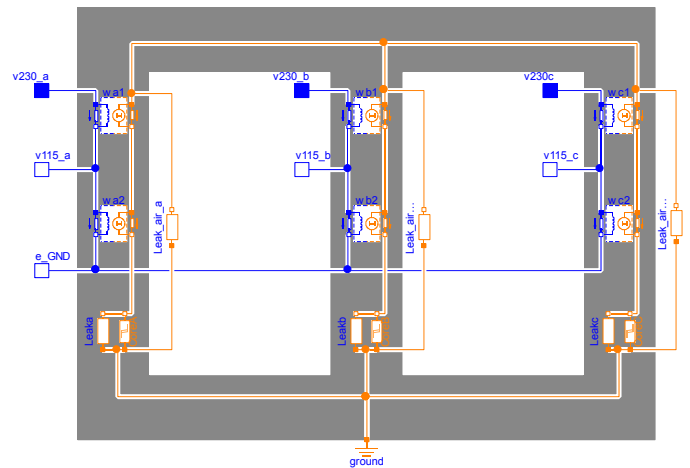


Figure 3: Modelica model of AC/AC auto-transformer

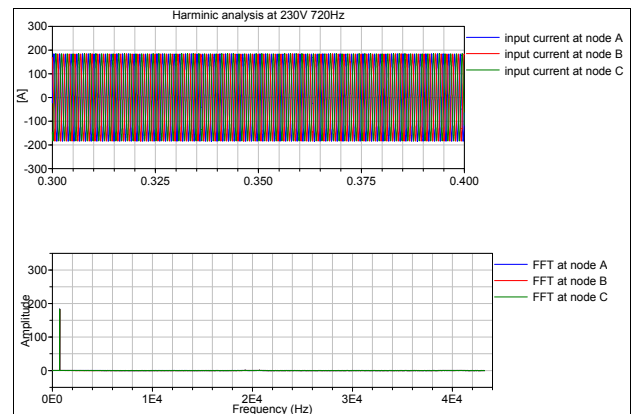


Figure 4: Current harmonic analysis of ATU at 230V and 720Hz input voltage

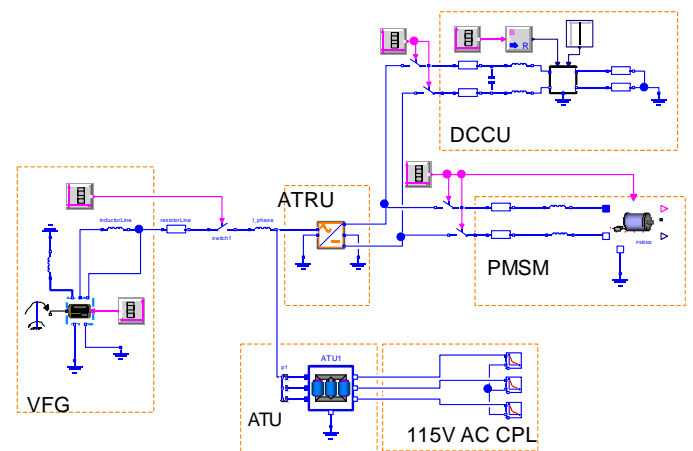


Figure 5: An integrated electric power network for MEA

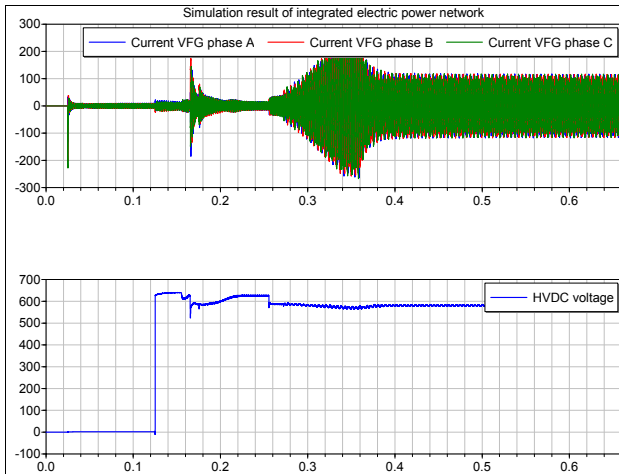


Figure 6: Simulation result of integrated electric power network: VFG current and HVDC voltage (behavioral level)

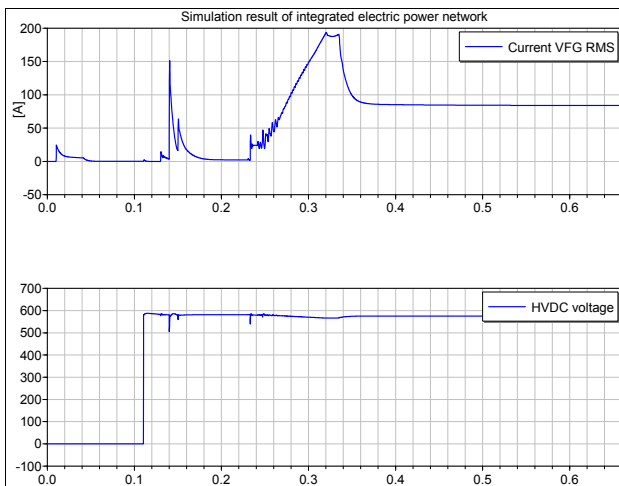


Figure 7: Simulation result of integrated electric power network: VFG current and HVDC voltage (functional level)

In practice the most common error is a stiffness of the equation system which can be traced back to modeling problems. Stiff means the eigenvalues of a linearized system have a real part which is negative and large in magnitude, compared to the reciprocal of the time span of interest [3, chapter 5]. For this problem usually no distinct warning is generated by the simulator. For small systems experienced designers may detect the sources of the problem by inspection. For larger systems built from verified components, new stability problems may arise while troubleshooting by inspection may fail due to the complexity.

The problem of modeling errors and its identification shall be illustrated via the circuit depicted in figure 8.

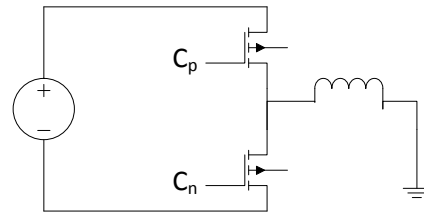


Figure 8: Example of switching circuit with inductance

The inductance on the right hand side is powered via the the transistors by positive or negative voltage. The most simple implementation of the transistor is a switching resistor with the levels high or low resistance, controlled by a Boolean input signal (e.g. Modelica.Electrical.Analog.Ideal.IdealClosingSwitch). For a pulse width modulation scheme, the inductor is powered by the upper or lower transistor where the control signal  $C_n$  is always the opposite of  $C_p$ . Since an overlapping operation of the transistors would result in a short circuit, for real systems the turn over of the complementary control signals includes a dead-time where both transistors are in high resistance mode. In simulation, a sudden interruption of the current path in the dead-time period results in a high voltage peak in the inductor due to  $v = Li$ . The inductor's flux vanishes slowly (stiff) due to the high switch resistances. This might sound like a simulation problem but in fact this is even a physical problem: For real transistors those voltages would also occur and could severely damage the semiconductor. This effect has to be prevented in design by a free wheeling diode or a "snubber circuit" in case of an inductive source. In the study this problem occurred for the motor inverter unit. As a counteraction the snubber circuit (a capacitance in parallel to the switch) was foreseen. While inserting tiny resistances/capacitances in the circuit is a well known but poor method to solve simulation problems, here the application adds physical meaning.

An other problem which can be related to physics is the "floating ground" problem. In the example, only the voltage difference of the voltage source is defined but the absolute levels of the connectors are calculated from the circuit. This may again result in chattering of the simulation if state variables are sensitive to this floating ground. The mathematical background of the problem is a ill-conditioned Eigenvalue vector and a high numeric sensitivity of the state. As in simulation "floating grounds" are problems of the real physical system as well which can lead to malfunctions of



the circuit. Both, circuit designer and model designer should be aware of it and should consider a direct connection to ground or a connection to ground via a large resistor. Floating ground problems often are shown in the simulator's translation log while the user may be overwhelmed by the amount of information at first.

For large systems there remains the problem of identification of the source of the stiffness. In the course of the study the following approach showed to be successful (Dymola specific):

1. DLR's Modelica Linear Systems 2 library is a very powerful tool to analyze a model. Since version 2013 its functions are embedded in Dymola's main window. The command "Full linear analysis" linearizes the system and gives the eigenvalues. A stiff system exhibits at least one large eigenvalue. The participation factors relate the states variables to the eigenvalues which identifies the critical components. In the example in figure 8, a large eigenvalue could be related to the inductor's state variable "current". The problem of the circuit can be detected by observing the current paths in the system.

Often the stiffness does not appear in the original system but in the course of the simulation. The "Full linear analysis" command can be invoked to linearize at a specified simulation time. For this, the next rules apply:

2. Simulation needs to start successfully. If the system does not initialize correctly a simulation from a no-load condition and successive connection of the loads may be successful. For Dymola, the "Dassl" solver generally shows a good compromise between accuracy and speed. For stiff systems there exist special solvers as "RadauIIa" (see [3, chapter 5] for example). In many cases the perpetual use of the solvers for stiff systems only conceals problems of the models which may return for slightly more complex systems. Nevertheless, for the identification of the problem it generally is a good idea to use a solver as "RadauIIa" to get the simulation running.
3. Simulation to the critical condition/time: The first step is to identify the critical condition by simulation. In many cases the critical time is the last successful simulation step. Then the "Full linear analysis" command is evoked with linearization at the critical time. Follow instructions 1..
4. If the simulation stops in the course of the simulation before an anticipated critical time or if no

evidence can be found at the last successful simulation step, a condition with similar critical conditions might be found in the earlier simulation. By observation of the "CPU time" (simulation time at each simulated time step) stiff conditions can be identified by steeper periods in the plot of the CPU time vs. simulated time.

## 5 New developments

In the course of the project, some deficiencies were identified for the implementation of the models and the automation of the results within a single environment. The following tools and scripts were newly developed to overcome the obstacles:

### 5.1 Signal processing tool

In an industrial design process, a tool chain must fulfill higher demands on automation, ergonomics and single tools are preferred than in usual research projects. Especially, tools for signal post operation are needed integrated in the design tool and must be scriptable. In JTI, Dymola was selected as single simulation platform and there was co-funding in the project MODELSSA to extend it by the necessary features. Amongst other features, a library of signal analysis methods in time domain (e.g. min/max, period, duty cycle, root-mean-square) and frequency domain (like FFT, IFFT, total harmonic distortion) was developed. All features are accessible graphically from the user interface and scriptable. The features are documented in [3] and were presented on the Modelica conference vendor sessions.

### 5.2 Modelling of magnetic hysteresis

For efficient modeling of magnetic circuits, the free Modelica.Magnetics.FluxTubes library [2] was applied for this study. The library is well established and was proven by hardware design studies. Material properties can be taken into account by linear and nonlinear permeability. As an important extension in the frame of the Cleansky project, DLR commissioned and supervised the development of magnetic hysteresis models for Modelica. As an outcome, a high fidelity model based on Preisach's equations was embedded (the original publication goes back to 1935, for implementation see [13] for example). For the study of inrush currents which was industrially motivated it was preferred to use another more efficient one. The so called "Tellinen hysteresis model" showed to

model the flux density  $B$  versus magnetic field strength  $H$  relationship of a measured ferromagnetic material well via moderate complex equations and thus by efficient simulation speed. Details and comparisons about the first release of the magnetic hysteresis models for Modelica were published in [17].

### 5.3 Dedicated scripting for tests

Most simulation tools provide basic post processing functions but with a limited perimeter. Thus, specific analysis functions usually have to be developed for the simulation platform using a post treatment language. Also systematic design and test automation often demands user specific scripts. When performing simple/single simulations, it is sufficient to select menu commands or to type commands in the command input line of the command window. But wanting to perform more complex actions as part of an industrial process (e.g. automatically repeat more complicated parameter studies a number of times) it is much more convenient to use the scripting facility. The goal is often to fully automate the simulation. Just to name some features, the script facility makes it possible to load model libraries, set parameters, set start values, simulate and plot variables.

Dymola supports easy handling of scripting, both with functions and script files (.mos files). Whether a function or a Modelica script file (.mos) should be used is up to the user, essentially the same functionality can be obtained with both. When a function should be the final result, a function is created, and the functionality is then created as an algorithm in this function using the Modelica Text layer of the function as an editor. When a Modelica script file (.mos) should be created, the command input line can be used for input, creating a command log that can be saved as a script. Scripts can be nested; functions can be nested and a Modelica script file may run other Modelica script files.

The test scripts were customized by DLR from existing commands of Modelica language and Dymola functions. The built in functions and model management tools of Dymola were found to be sufficient for the study. Among others, the following scripts and tools were found to be necessary repeatedly for the V&V study:

- Transformation of input data: From input files data are edited into a Modelica compatible type. This function is of major relevance in an industrial process since input data might not be Modelica compatible and use of additional software

is undesirable in standardized processes. By help of some Dymola built in functions conversions from text files, Microsoft Excel sheets and more was performed. Matlab's matrix data format .mat is supported immanent.

- Transformation of output data: Same as for the import, the output features are important. Typical outputs programmed in functions were time domain results (.mat result files), tables (Excel) and generic data (.txt). For generation of frequency domain data, it was necessary to have the Fast Fourier Transformation (FFT) function executable by script. Thanks to the co-funding by the JTI project this function is now provided in Dymola. Application of the FFT function include parameter studies with tabulated output of total harmonic distortion (THD) or harmonic content at specified harmonics.
- Coherency test models: Test of linear dependence of signals in time domain by convolution. Coherency is an important criterion in verification tests to analyze the validity of models and model abstraction levels. It is applied for verification of models versus hardware test data or between different models. The script calculates coherency by application of the coherency function to two simulation or measurement waveforms.
- AC modulation envelope: The AC envelope function is the smooth curve outlining the extreme positions of a distinctive alternating wave with a fixed frequency. The modulation envelope shows its amplitude variation in frequency domain. The AC amplitude is an important measure for the voltage quality which must be stabilized by the generator control. The developed function relies on peak finding and transformation of data to frequency domain by FFT.

As an example of an integrated test, the Dymola script for the worst case study of sympathetic effect is presented in Fig. 9. This script function firstly simulates the test bench till 5.791 second and records the simulation result in a "end.txt" file. This result will be always defined as initial condition for the following 20 time simulations with different connecting time for ATU2. A real variable Tatu2 is defined to vary the connecting time of ATU2 and has a time step  $\varepsilon = 0.000125$  second for each simulation loop. Simulation setups such as used integrator, simulation time are also defined in the script function. Additional scripts have been made

for post-processing. It is a big advantage for users, that a complex test like analysis of sympathetic effect can be easily formulated with Dymola scripting language in a very compact manner.

```
function Script_Case1_with_simulation_paper
import Modelica.Utilities.Flies.*;

protected
  Real Tatu2 = 5.791;

algorithm

  //performe simulation till steady-state
  translateModel("Case1");
  simulateModel("Case1",
    stopTime=5.79,
    numberOfIntervals=0,
    method="Radau",
    resultFile="Case1");

  //backup dsfinal
  copy("dsfinal.txt","end.txt",true);

  //do parameter studies about
  //switch on time of ATU2 with
  //time step 0.000125s

  for i in 1:1:20 loop

    Tatu2 :=5.791 + i*0.000125;
    importInitial();
    simulateExtendedModel("Case1",
      startTime=5.79,
      stopTime=5.8,
      numberOfIntervals=0,
      method="Radau",
      resultFile="dres_"+String(i),
      initialNames={"Tatu2"},
      initialValues={Tatu2},
      finalNames={"Tatu2"});

    //re-initialization
    Files.copy("end.txt","dsfinal.txt",true);

  end for;
end Script_Case1_with_simulation_paper;
```

Figure 9: Scripting for the worst case study of sympathetic effect

## 6 Conclusion

By the study, the applicability of Modelica and Dymola for large scale testing of aircraft electrical systems in V&V studies was demonstrated successfully. All demands on new functionality, additional models and specialized scripts could be met within the project.

It could be detected the reliability of simulation highly depends on mature models. The developed library therefore is an important base for propagation to a Modelica based V&V process in aircraft electrical systems simulation. Further effort should be made for robust initialization of the simulation.

## Acknowledgment

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) for the Clean Sky Joint Technology Initiative under grant agreement no. CSJU-GAM-SGO-2008-001.

## References

- [1] J. Bals, Y. Ji, M. Kuhn, and C. Schallert. Model based design and integration of more electric aircraft system using modelica. *Moet Forum at European Power Electronics Conference and Exhibition*, 2009.
- [2] T. Bödrich and T. Roschke. A magnetic library for modelica. In *Proceedings of the 4th International Modelica Conference*, 2005.
- [3] Dymola. User manual volume 1. Technical report, Dassault Systèmes AB, 2013.
- [4] CleanSky project: Systems for Green Operation (SGO). <http://www.cleansky.eu/>.
- [5] T. Giese, D. Schlabe, R. Slate, M. Crespo, F. Tichy, and C. Baumann. Extended design office concept definition. *Cleansky WP2.1.1 deliverable*, 2010.
- [6] Y. Ji and M. R. Kuhn. Physical modeling and simulation of inrush current in power transformers of more electric aircraft. In *IEEE IFEEC conference*, 2013.
- [7] Y. Ji and M.R. Kuhn. Modeling and simulation of large scale power systems in more electric aircraft. In *IEEE 14th Workshop on Control and Modeling for Power Electronics (COMPEL)*, pages 1–6, 2013.
- [8] Y. Ji, A. Pfeiffer, and J. Bals. Optimization based steady-state analysis of switched power electronic systems. In *Control and Modeling for Power Electronics (COMPEL), 2010 IEEE 12th Workshop on*, pages 1–6, 2010.
- [9] Paul C. Krause, Oleg Wasynczuk, and Scott D. Sudhoff. *Analysis of electric machinery and drive systems*. Wiley-Interscience, Piscataway, New York, 2nd edition, 1998.

- [10] M. Kuhn and M. Otter. A multi level approach for aircraft electrical systems design. *6th International Modelica Conference*, 2008.
- [11] M.R. Kuhn, Y. Ji, H. D Joos, and J. Bals. An approach for stability analysis of nonlinear electrical network using antioptimization. In *Power Electronics Specialists Conference, 2008. PESC 2008. IEEE*, pages 3873–3879, 2008.
- [12] M.R. Kuhn, Y. Ji, and D. Schröder. Stability studies of critical dc power system component for more electric aircraft using  $\mu$  sensitivity. In *Control Automation, 2007. MED '07. Mediterranean Conference on*, pages 1–6, 2007.
- [13] I. D. Mayergoyz. *Mathematical Models of Hysteresis and Their Applications*. Elsevier, 2003.
- [14] C. Schallert. Inclusion of reliability and safety analysis methods in modelica. *8th International Modelica Conference*, 2011.
- [15] Daniel Schlabe, Michael Sielemann, Christian Schallert, Dirk Zimmer, Martin Kuhn, Yang Ji, and Johann Bals. Towards a model-based energy system designprocess. In *SAE Power Systems Conference 2012*, Oktober 2012.
- [16] Ji. Y, J. Bals, and A. Pfeiffer. Multi-level power quality assessment towards virtual testing of more electric aircraft. In *IPEC, 2010 Conference Proceedings*, pages 28–33, 2010.
- [17] J. Ziske and T Bödrich. Magnetic hysteresis models for modelica. *9th International Modelica Conference*, 2012.

# Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters

Franciscus L.J. van der Linden  
German Aerospace Center (DLR),  
Institute of System Dynamics and  
Control, Münchner Straße 20  
82234 Weßling, Germany  
Franciscus.vanderlinden@dlr.de

Clemens Schlegel  
Schlegel Simulation GmbH  
Meichelbeckstr. 8b  
85356 Freising, Germany  
cs@schlegel-simulation.de

Markus Christmann  
EADS Innovation Works, TCC6  
81663 Munich, Germany  
markus.christmann@eads.net

Gergely Regula  
MTA SZTAKI  
13-17 Kende St.  
1111 Budapest, Hungary  
gergely.regula@sztaki.mta.hu

Chris I. Hill  
University of Nottingham  
Aerospace Technology Centre,  
Innovation Park  
Nottingham. NG7 2TU. UK  
c.hill@nottingham.ac.uk

Paolo Giangrande  
University of Nottingham  
Tower Building,  
University Park  
Nottingham. NG7 2RD. UK  
p.giangrande@nottingham.ac.uk

Jean-Charles Maré  
Institut National des Sciences Appliquées  
135 Avenue de Rangueil  
31077 Toulouse Cedex 4, France  
jean-charles.mare@insa-toulouse.fr

Imanol Egaña  
IK4 TEKNIKER  
Calle Iñaki Goenaga, 5  
20600 Eibar, Gipuzkoa, Spain  
imanol.egana@tekniker.es

## Abstract

The goal of the A2015 library presented in this paper is to develop a Modelica based, tool-independent standard for electromechanical actuators (EMA). This will contribute to the establishment of a “common language” throughout the development of EMAs for aircraft and helicopters and through the supply chain. All stages of the design and validation process (conceptual design, specification, development and validation) are covered. The modeling approach addresses specific aspects of the EMA design process not covered by existing tools. The library scope, engineering need and implementation are described. Modeling of selected EMA components is discussed in more detail. An application example of the library is given (linear actuator, A320 aileron)

*Keywords: Actuator, EMA; Library; Multi Physics*

## 1 Library scope

### 1.1 Introduction

Protecting the environment and providing efficient onboard energy supply is one of the top goals in the

development of future aircraft. A key technology towards the realization of these goals is the “More Electric Aircraft”. Various research initiatives have been launched in recent years to get closer to this goal. The ACTUATION2015 project [1] will complete this approach by focusing on Electro Mechanical Actuator (EMA) technologies. EMAs are mandatory in order to substitute hydraulic circuits, pumps and reservoirs. The objective of the ACTUATION 2015 project is to develop and validate a common set of standardized, modular and scalable EMA modules that address cost, reliability and weight requirements from the air framers. In the context of that project the A2015 simulation model library is developed to support and streamline the EMA design process.

### 1.2 Engineering needs

The focus of the ACTUATION2015 project is on aircraft families including business, regional and large commercial aircraft as well as helicopters. The actuation systems covered include primary and secondary flight controls, main landing gears and breaks. A further major focus is on modularization and scalability of EMA modules. All the needs and architecture differences of such systems and all dif-

ferent stages of the development process (conceptual design, specification, development, and validation) through the whole supply chain must be covered by a universal simulation model library.

Apart from covering the relevant physical effects and general system dynamics aspects, the library should support the development in terms of concept and performance assessments, sizing (as far as system aspects are involved), component requirements definition and fail case assumptions, system reaction and performance in case of failures (static and transient), and virtual design validation. Electromechanical actuators have mostly a reduced reliability compared with conventional hydraulic actuators. Therefore the library must allow assessing redundant actuator concepts.

Since simulations on system and subsystem level can't cover all computational needs of an EMA development, interoperability with established tools is vital. Because many tools support it, the FMI standard [13] is the natural choice for model exchange of Modelica libraries. The library does not provide distributed parameter models (for e.g. stress computations) and monitoring and state of health algorithms, because specialized tools for these tasks are already available.

### 1.3 Library architecture

In order to cover the described range of applications several models of different scope and level of detail are implemented for each of the core EMA components (multi-level approach, see e.g. [14]). Since each group of models shares the same interface they are easily replaceable. In the A2015 library five modeling levels are predefined, mostly associated with nonlinearities included and events triggered (for easier handling each level has an associated icon color):

1. Perfect (linear, no losses)
2. Linear, invertible
3. Nonlinear, invertible (e.g. using tanh instead of sign functions)
4. Hard nonlinear (state events are triggered by nonlinearities)
5. Fully switched (e.g. switching inverter, based on state events)

This scheme does not imply that for all components models of all levels are included, or for each level only a single model is included. Instead, the modeling level scheme is introduced to give the user a quick idea what kind of effects a certain model does include.

Naturally, the A2015 library follows a standardized model breakdown structure with common interfaces based on the Modelica Standard Library (MSL) and uses its predefined implementations of thermal flows and heat exchange, and mechanical support flanges [9]. Parameters are organized in records, signals are bundled in busses. In order to unify failure injection, a dedicated library is used [12].

## 2 Selected library components

The A2015 library contains model components from the following domains: Electrical (inverters, motors), mechanical (rotation to rotation and rotation to translation transformers), sensors (position, speed, force, etc.), thermal (heat sinks, housings), and control (e.g. force fight compensator). Selected components are described in this paragraph.

### 2.1 Fault Triggering Library

The development of health monitoring algorithms, but also the design of fault tolerant redundant actuators is aided by the possibility to easily trigger faults in the models.

The `FaultTriggering` library [12] is used as a standardized approach to trigger all failures implemented in the library. At the moment the motor, inverter, gear and nut screw models have predefined faults that can be enabled. Further model components will follow.

### 2.2 Inverters

This section gives an overview of the implementation of the multi-level Power Electronic Inverter modeling provided within the A2015 library. For further technical details see [3].

#### 2.2.1 Package Structure

As described previously, a multi-level modeling approach is used within the A2015 library. For the Power Electronic Inverters 5 levels of complexity are provided. The Inverters package is split into the `NonSwitching` and the `Switching` subpackages, see figure 1. The `NonSwitching` package provides models for levels 1 to 4 and the `Switching` package provides a level 5 model. A core feature of the A2015 library is that for each individual component every modeling level is fully replaceable with one another. In order for each modeling level within the Inverters library to be fully exchangeable a common interface is used for all 5 modeling levels.

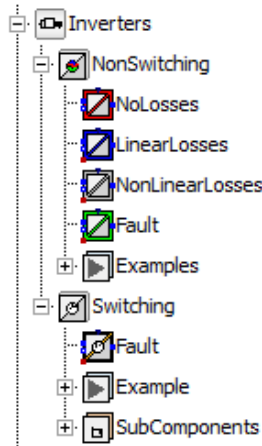


Figure 1: Structure of the Inverters package as part of the Actuator Library.

### 2.2.2 Losses

It is important to note that all the Power Electronic Inverter models are fully multi-directional. Therefore losses are fully implemented, regardless the direction of the power flow. Hence, if power flows from DC to AC the AC side power will be lower than the DC side power and vice versa. This is inherent within the model and will be calculated automatically even when the direction of power flow changes during simulation. Table 1 summarizes the losses included in each level of the Inverter models.

Level	Losses
1	None
2	Linear Losses
3	Non-Linear Losses
4	Non-Linear Losses
5	Conduction Losses

Table 1: Losses modeled within each level of the Inverters package.

Using the level 2 model the user may specify a constant efficiency for the Power Electronic Inverter under all operating conditions (a default value is provided, if no data is available). Levels 3 and 4 both use non-linear loss characteristics. The user can specify the power range of the inverter and the losses within the inverter over the specified power range. A default characteristic is also given, as shown in figure 2 below, however accurate characteristics are recommended to be added by the user if known. The default characteristics are per unit and are scaled within the model by the specified, or default, power range.

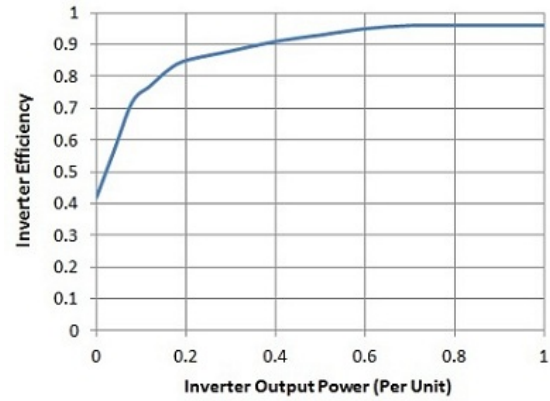


Figure 2: Default non-linear loss characteristics

Level 5 includes switches and Pulse Width Modulation in order to create the required electrical outputs. The losses included are the conduction losses due to the switch and diode resistances when conducting. Energy losses due to switching are neglected.

### 2.2.3 Faults

Another core aspect of the A2015 library is the inclusion of fault conditions within all component models. In the case of the Power Electronic Inverters these are mainly included in the level 5 model because failure of switches are the most likely failure conditions and no switches exist within the lower level models. Table 2 summarizes the fault conditions included in the Inverter models.

Level	Faults
1	None
2	None
3	None
4	Full-Bridge
5	Single Switch, Single Phase, Multi-Phase and Full Bridge

Table 2: Fault conditions modeled within each level of the Inverters package.

For level 4 a full bridge fault is implemented. When triggered the Power Electronic Inverter output becomes zero. This represents a full-bridge open circuit fault or deactivation of the Inverter. The level 5 model includes the ability to activate open circuit and short circuit faults. These can be injected for a single switch, multiple switches, a single phase, multiple phases and full bridge. The Fault Triggering library described in [12] is used in order to trigger the implemented faults. The actual faults can be injected at any time instance but must be predefined.

## 2.3 Motors

### 2.3.1 Package Structure

According to the machine topologies considered, the Electrical Machines package is divided in four sub-packages:

- Basic, which considers a generic AC machine
- PermanentMagnet, which includes the permanent magnet synchronous machine (PMSM) models
- Reluctance, which models synchronous reluctance machines (SRM)
- DirectCurrent, which contains the models of DC machines with independent excitation

An Example package is provided for each of the described sub-packages, in order to highlight the model features. The Electrical Machines package has been developed adopting a multi-level approach as described in chapter 1. Several model levels of the same machine type, with different physical effects taken into account have been implemented.

### 2.3.2 Model Interfaces

The interchangeability among model levels is ensured using common interfaces. Considering AC electrical machines `PositivePlug` and `NegativePlug` have been used as electrical interface. In case of DC machine models, `PositivePin` and `NegativePin` have been adopted as electrical interfaces, for both armature and field windings. All the machine models adopt `Flange` and `Support` as mechanical interfaces and contain a heat port providing housing temperature and heat flow.

### 2.3.3 Basic Model

The basic model is the simplest model for a generic AC machine. The machine behavior is described by a torque constant, which establishes the relationship between current and electromagnetic torque. Losses, non-linear effects and thermal behavior are not considered at this modeling level. The basic model, as all the other machine models, can work reversibly representing both motor and generator.

### 2.3.4 Standard Models

Standard models are based on a lumped parameter approach. They describe the behavior of the machine considering losses and thermal behavior, but neglecting magnetic saturation, torque ripple and fault conditions. Standard models have been developed for PMSM, SRM and DC machine. These models are power balanced. The losses taken into account are:

copper, iron [4], mechanical and for DC machine brush losses. Finally, standard models provide a machine thermal model, which gives the winding and housing temperatures, and the heat flow.

### 2.3.5 Saturation Models

Saturation models for PMSM, SRM and DC machines have been obtained by extending the standard models. The main features of the Saturation models are magnetic saturation and torque ripple. Magnetic saturation is taken into account using two approaches (which can be user-selected): look-up table and analytical implementation. In the former case, the torque-current curve data (may be obtained by a finite element simulations) are stored in a look-up table, within the model. In the latter case, the torque-current curve is implemented using three given points (current and torque at the knee and torque at the saturation region). Finally, Saturation models allow superimposing a sinusoidal torque ripple on the computed electromagnetic torque. The torque ripple parameters (amplitude and frequency) must be given by the user.

### 2.3.6 Fault Models

Models including fault conditions have been implemented only for PMSM, since the presence of permanent magnets (PMs) is source of concerns in the case of failures. Fault conditions such as winding short-circuits, winding open-circuits and PM demagnetization have been considered. For each of these cases a specific PMSM model has been developed in order to minimize the model complexity. The fault models are:

- ShortCircuit, which implements single- and three-phase short-circuits
- OpenCircuit, which considers single- and three-phase open-circuits
- Demagnetization, which takes into account the PM demagnetization.

As suggested in [5], the winding faults have been implemented by means of ideal electrical switches. In the demagnetization model, the fault condition is taken as a consequence of the fault (i.e. PM flux reduction) neglecting the causes that led to the fault.

## 2.4 Mechanical rotational (reducers)

The rotational mechanical models include gear reducers and torque limiters. As with the other components, multiple modelling levels are included for different accuracy needs.



### 2.4.1 Package Structure

The package structure is given in figure 3. Only selected packages are expanded to give a good overview of the used models.

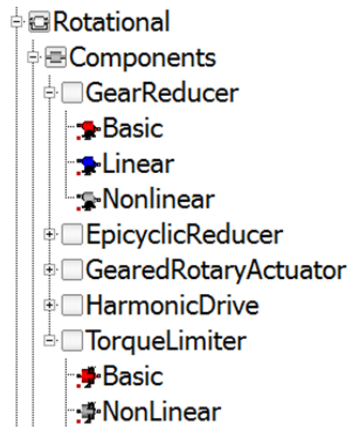


Figure 3: Structure of the Rotational package.

### 2.4.2 Model properties

All rotational models are fully balanced models with optional support flange. Using this approach, the support torques can be modelled and used for housing design. Also all models are energetically balanced. All losses are available in the (optional) heat ports.

### 2.4.3 Gear reducers

The `GearReducer`, `EpicyclicReducer`, `GearedRotaryActuator` and `HarmonicDrive` packages model geared transmissions with following levels of detail:

- Basic: no losses
- Linear: losses dependent on speed
- Nonlinear: losses dependent on the speed using a hyperbolic tangent approach to approach the nonlinear friction effect at zero velocity.

### 2.4.4 Torque limiter

The torque limiter restricts the torques on a shaft to a safe level by transferring the rest torque to a support flange.

## 2.5 Mechanical translational (nut screws)

As illustrated by table 3, the most simple models of nut screws (NS) start from the MSL `IdealR2T` model to reproduce the functional effect of perfect power transformation between rotational and translational domains.

For the most simple case, end-stops and bearings are not explicitly considered in the NS model:

- When the displacement is out of physical stroke, a warning is generated or the simulation is stopped.
- Anti-rotation and anti-translation devices are included implicitly, assuming no translation of the drive shaft and no rotation of the actuator rod.

When using the support options, the NS model becomes a mechanical quadriport model which combines the rotation and translation of nut and screw. This enables the explicit modelling of anti-rotation and anti-translation outside the NS model [6]. Inertial effects are also to be added outside of the NS model.

Technological imperfections are progressively introduced and modelled with increasing complexity. Among all candidate topologies, the NS model structure is selected in order to meet major engineering needs for aerospace actuators. For this reason, it associates in series (from drive to load flanges):

- a perfect rotational to translational power transformer, with warning or error generated when the position is out of stroke
- an optional friction loss that applies to a rigid rod
- an optional and generic compliance that can represent in addition either backlash or preloading, depending on parameterisation.

### 2.5.1 Friction models

There are a lot of candidate options for friction modelling and parameterization [7]: efficiency (direct/indirect), parametric model of friction force (e.g. Coulomb or Stribeck), coefficient of friction with rod diameter and lead, etc. Using efficiency (direct and indirect) has been selected for non linear friction models because it is widely used by engineers. As a drawback, using efficiency exclusively does not allow to compute the no-load driving force or no-drive back-driving force. For non preloaded designs, a lossy rod model has been developed with the same structure as the MSL `LossyGear` [8].

For continuous models, the transition between sticking and sliding has been approximated by a hyperbolic tangent function in order to avoid triggering state events. In addition, removing any conditional statements has contributed to typically reduce the simulation time of the friction model by 25%. By setting negative values for the indirect efficiencies, irreversibility (impossible to back drive the NS) is parameterized without introducing an additional pa-

parameter for that case. When activated the thermal port of the friction model outputs the heat generated by friction.

### 2.5.2 Compliance models

In order to enable failure injection or design exploration, a non-linear model has been developed to reproduce backlash, pure compliance or preloading, depending on its parameterization. NS compliance has been modelled as a transmission device involving two unilateral elastic contacts that act in opposition. The compliance model structure was defined in order to maintain consistency with the MSL, as illustrated by table 3.

The MSL `Elastogap` model has been modified for non-linear continuous models in order to avoid any event triggering or conditional statements, even for structural damping. Each direction of contact between nut and screw has been modelled by a serial combination of a lossy-rod and an elasto-gap, with one elasto-gap being flipped. This choice has the major advantage of reproducing the friction force

due to preload without any modification of the friction model. For non continuous models, the strengthening length of contacts has been used as the unique parameter of preload and backlash, enabling continuous transition between these characteristics. Positive values correspond to backlash, zero values to pure elastic and negative values to preload conditions.

### 2.5.3 Fault triggering

The two major types of NS failure are jamming and free-run. Jamming has been modelled by increasing friction through modification of the friction model parameters. Free-run has been modelled by manipulation of the compliance model parameters: decreasing stiffness (linear model) or preload, then increasing backlash. The failure injection library can also be used to introduce progressive degradation through continuous evolution of friction and compliance parameters. This is of particular interest for virtual assessment of health monitoring strategies.

Properties of different nut screw models	MSL	Basic	Linear	Nonlinear Continuous	Nonlinear non continuous
Perfect power transformation					
Optionally activated ports of mechanical support	✓	✓	✓	✓	✓
Replaceable models and data records					
Warning or stop if out of stroke		✓	✓	✓	✓
Offset of displacement transformation		✓	✓	✓	✓
Optionally activated heat port	✓	✓	✓	✓	✓
Optional version for Failure injection (jamming or free-run)		✓	✓	✓	✓
Compliance			Linear spring	✓	✓
Friction loss	Direct/Inverse efficiency		Viscous damper	Direct/Inverse efficiency	Direct/Inverse efficiency
Irreversibility				✓	✓
Preload				✓	✓
Backlash				✓	✓
Effect of velocity on friction				Planned	Planned
Effect of temperature on friction				Planned	Planned

Table 3: Properties of A2015 library nut screw models

### 2.5.4 Model verification

All models have been designed to accept implementation in causal simulation software. In addition, their ability to run correctly for any causality case in the Modelica environment (any combination of two imposed variables out of drive torque, drive velocity, load force, and load velocity) has been addressed in detail.

### 2.5.5 Future work

The last part of the model development will include the consideration of the dependency of friction on velocity and temperature. Unfortunately, these effects are rarely documented in supplier's catalogues and are generally considered as confidential. However, it is intended to enable varying efficiencies versus velocity and temperature, either through a parametric equation or by a 3-D table.

## 2.6 Sensors

The A2015 library includes models for linear and angular position sensors, force sensors, torque sensors, current sensors, and temperature sensors.

They extend ideal sensor models from the MSL [9] and add effects and failures to the sensor output that are present in real sensor devices. Offset, non-linearity, saturation, and temperature dependency, as well as hysteresis, drift, and signal discretization are the effects taken into account. Most common sensor failures, such as open-circuit and short-circuit, will also be implemented.

The sensor models are parameterized according to typical data sheets in order to provide the end user with a practical library. This approach results in very general sensor models, independent from the sensor technology, which can be easily customized.

Figure 4 shows the structure of the Sensors package, which follows the general modeling approach for the whole library: different model levels (here: `Ideal`, `Linear`, and `NonLinear`) sharing a common interface, parameters included in records, and a bus connector predefined with the ideal and real signals of the sensor, and a status flag.

## 2.7 Controllers

### 2.7.1 Package Structure

This package contains blocks suitable for designing the control system of actuators [10]. The blocks can be combined with actuators of different modelling levels.

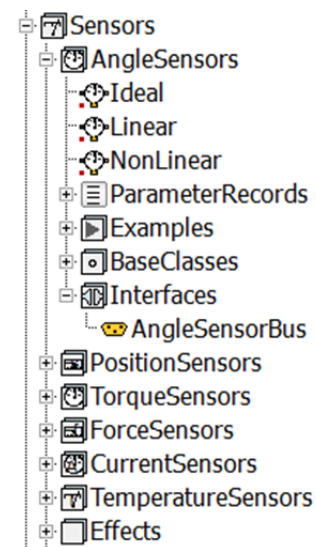


Figure 4: Structure of Sensors package

The structure of the Controllers package is shown in figure 5. Due to the function of the models within the package, the structure of the Controllers package follows a slightly different approach than other packages of the A2015 library.

The package consists of four sub-packages. The Continuous and Discrete sub-packages contain basic controller blocks built up from proportional, integrator and derivative blocks. The main purpose of the blocks in the Filters sub-package is to smooth control demand signals or filter measurements. The last sub-package contains blocks performing direct and inverse Park transformations. The package contains wrapper classes that allow the user to conveniently exchange blocks with similar functionality. The blocks in the library are parameterised via dedicated parameter records, organized in a separate sub-package.

### 2.7.2 Description of selected blocks

The PID controller variants include a symmetrical output saturation and integrator anti-windup (when applicable) and can be parameterised according to both the serial and parallel convention. They differ from the counterpart in the Modelica Standard Library in the implementation of the integrator anti-windup. Within the A2015 library, the integrator anti-windup is based only on the output of the integrator, not on the overall output. The output saturation is therefore implemented separately.

The filter blocks include both continuous and discrete time blocks, as well as general blocks that can be parameterised by specifying the numerator and

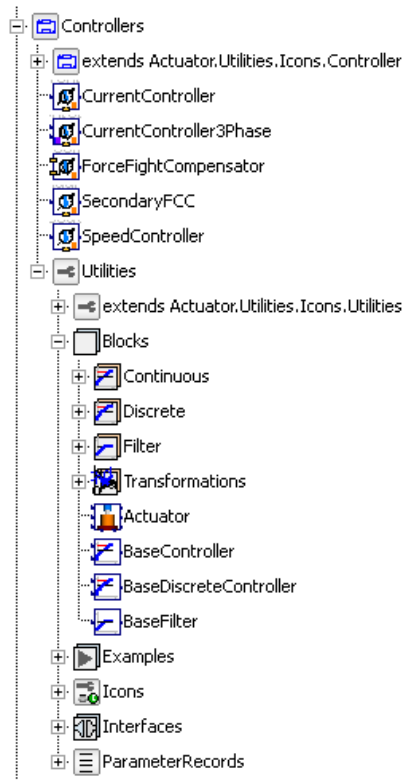


Figure 5: Structure of the Controllers package.

denominator of their transfer functions and also the well-known continuous time filter types available in the Modelica Standard Library (Butterworth, type I Chebyshev, etc). These model classes are wrappers for the blocks from the Modelica Standard Library in order to allow the user to switch between the different filter types.

A specialised force-fight compensator, the position, speed and current controllers are available as distinct standalone blocks. The measurements required by these blocks are collected via specialised sensor buses (see chapter 2.5). The user is free to select the controller and control demand filter types. These blocks can be connected to motor models via a three-phase interface and also to simplified motor models with one input. This is used in the two variants of the current controller, one of which performs space vector control.

### 2.7.3 Failure modes

The actuator control library also provides the user with a means for simulating the effects of various control failure scenarios. The following failure modes will be implemented: control output freeze, controller reset, short spikes in the controller output, non-return to zero error, runaway and no valid data faults.

## 3 Application example

To test the library and give a realistic example, a linear actuator based on an A320 aileron actuator has been modeled using the described library. The actuator model is set up using the base class `LinearGearedEMA`. This is a predefined base class for linear actuators which fully supports all redeclaration schemes of the library. It is extended by a position controller and an extra position sensor.

The final model can be seen in figure 6. In the model, the multi disciplinary approach of the library is directly visible. Electrical, mechanical and thermal effects are taken into account as well as the actuator control.

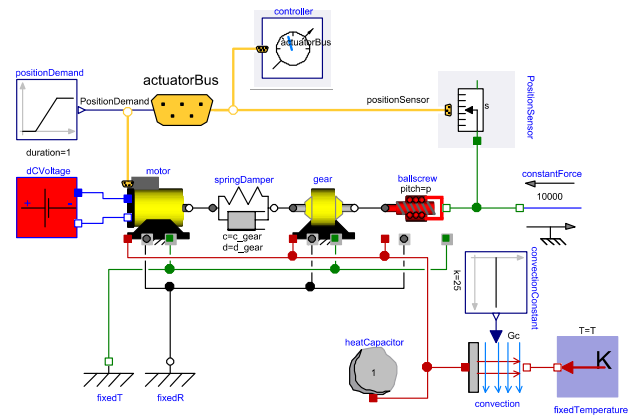


Figure 6: Overview of complete Linear EMA

### 3.1 Model structure

The mechanical model is built using a 2x1DOF approach. The rotational as well as the translational forces and support forces generated by each part are accessible outside the model. This may be used for the design of anti-rotation devices. The parameter data of the motor, gear, inverter, nut screw and controller are stored in parameter records. They extend from base records of these data to facilitate easy re-declaration in the models.

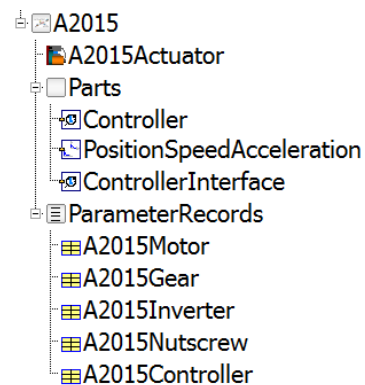


Figure 7: Overview of the library with the components for a specific linear EMA

In figure 7 an overview of the library for this specific linear EMA is given. All specific parts are combined in the package `Parts` and all parameters in the package `ParameterRecords`.

### 3.2 Simulation results

Simulation results of the described actuator can be seen in figure 8. The shown current response is the result of a position step input of 4.3 mm. In the Modelica model, the aerodynamic damping is neglected and a continuous controller is used. For comparison the simulation result of a discrete Simulink reference model is also plotted. Both simulations show a comparable dynamic response.

By modelling the controllers using the existing discrete blocks, it is possible to also model the effects of discrete controllers and increase the match between the model results.

To showcase the more advanced features of the library, an advanced model, including motor saturation and motor ripple is included. The effect of a motor ripple is visible from 1.15 to 1.3 seconds.

The shown simulation run takes 0.2 seconds of simulation time on an Intel® Xeon® E5-1620 Processor. The reference model in Simulink takes 6.5 seconds simulation time on the same machine. Including a sampled model using the same sample time as the Simulink model increases the simulation time to 1.7 seconds.

The frequency response of an actuator is also an important design criterion. It can be calculated using the `FrequencyResponse` package [11] by T. Bunte. The amplitude and phase response for 3 amplitude values (1, 5 and 10 mm) is calculated. The response from the input reference to the output rod is shown in figure 9. The frequency response of the same system with different amplitude changes depending on the stroke. This clearly shows the nonlinearity of the system. As expected, the system cannot follow the high frequencies at high amplitudes due to limitations on the motor current.

## 4. Conclusion and outlook

In this paper we have presented a library for modeling Electromechanical Actuators (EMAs). In the library, electrical machines, inverters, controllers, mechanical parts (rotational and translational) as well as sensors have been included. All model components have been set up using replaceable parameter sets and replaceable parts. The standardization of the interfaces of the components using partial classes

enables the user to build an EMA model as a modular system. This supports the exchange of models between partners and makes it possible to reuse these models for different simulation tasks without a structural redesign. The integration of component faults helps to quickly assess the actuator performance for failure cases.

To test the library, an existing Simulink model has been reworked into a Modelica model using mostly the components from the library. The results of the Modelica model show a good agreement with the Simulink reference model.

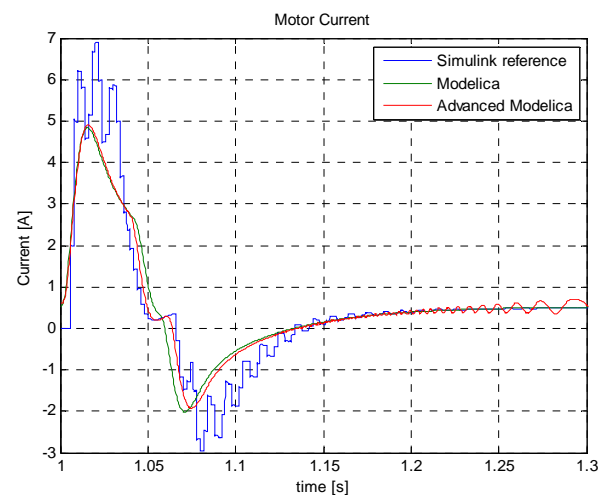


Figure 8: Simulation of actuator system

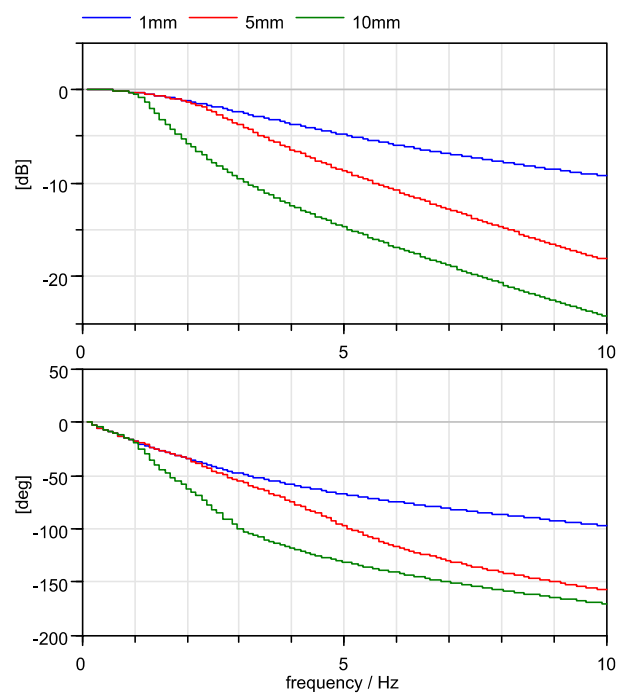


Figure 9: Frequency response of controlled actuator from position reference input to rod output.

The presented library is developed with a strong focus on Electromechanical Aircraft and Helicopter Actuators. This is not the only field the library can be used; It can help developers of all electromechanical actuators like automotive actuation (air path actuators for diesel engines, automatic hatch door actuation), automation systems (packaging machines) or printing systems.

Possible future library extensions include mass, volume, cost, etc. design optimization (by providing specialized modeling levels while maintaining the EMA model topology) and automated safety and reliability analysis.

The availability of the library in- and outside of ACTUATION 2015 is currently under clarification by the partners of ACTUATION2015.

## Acknowledgement

The research leading to these results has received funding from the European Union's Seventh Framework Program (FP7-284916) for ACTUATION 2015 under grant agreement no. 284915.

## References

- [1] [www.actuation2015.eu](http://www.actuation2015.eu)
- [2] [www.modelica.org](http://www.modelica.org)
- [3] C.I. Hill, P. Giangrande, C. Gerada and S.V. Bozhko, Implementation of a Multi-Level Power Electronic Inverter in Modelica. In *proceedings of the 10<sup>th</sup> Modelica Conference*, 2014.
- [4] N. Urasaki, T. Senjyu and K. Uezato, A Novel Calculation Method for Iron Loss Resistance Suitable in Modeling Permanent-Magnet Synchronous Motors, *IEEE Transactions on Energy Conversion*, vol. 18, no. 1, pp. 41-77, March 2003.
- [5] D. Winkler and C. Gühmann, Modelling of Electrical Faults in Induction Machines Using Modelica, *Proceedings in 48th Scandinavian Conference on Simulation and Modeling (SIMS)*, 2007.
- [6] Maré J-C., 2-D Lumped parameters modelling of EMAs for advanced virtual prototyping of EMAs, *Proceedings of Recent Advances in Aerospace Actuation Systems and Components*, pp 122-127, Toulouse, France, June 13-14, 2012
- [7] Maré J-C., Friction modelling and simulation at system level: a practical view for the designer, *ImechE part I, Journal of Systems and Control Engineering*, Volume 226 Issue 6, pp. 728 - 741, July 2012
- [8] Otter M., Elmqvist H., Mattson S. E., Hybrid modelling in Modelica based on the synchronous data flow principle, *1999 IEEE Symposium on Computer-Aided Control System Design*, pp. 151-157, August 22-26 Hawaii, USA, 1999
- [9] MSL: Modelica Standard Library, [www.modelica.org/libraries](http://www.modelica.org/libraries)
- [10] I. Réti, M. Lukátsi, B. Vanek, I. Gőzse, Á. Bakos, J. Bokor, Smart mini actuators for safety critical unmanned aerial vehicles, *2nd International Conference on Control and Fault-Tolerant Systems (SysTol)*, Nice, France, 2013.
- [11] Bünthe, T., Recording of Model Frequency Responses and Describing Functions in Modelica. In: *Proceedings of the 8<sup>th</sup> International Modelica Conference*, Technical University Dresden, Germany (pp. 686–696), 2011.
- [12] Linden, F. L. J. Van der, General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. In *proceedings of the 10<sup>th</sup> Modelica conference, 2014*
- [13] Functional Mock-up Interface [www.fmi-standard.org](http://www.fmi-standard.org)
- [14] M. Kuhn, M. Otter, and L. Raulin. (2008): A Multi Level Approach for Aircraft Electrical Systems Design. In: *Proceedings of the 6<sup>th</sup> International Modelica Conference*, Bielefeld, Germany, pp. 95-101, 2008

# An Optimization Framework for Dynamic Hybrid Energy Systems

Wenbo Du<sup>1</sup> Humberto E. Garcia<sup>1</sup> Christiaan J.J. Paredis<sup>2</sup>

wenbo.du@inl.gov humberto.garcia@inl.gov chris.paredis@gatech.edu

<sup>1</sup>Idaho National Laboratory  
750 University Boulevard, Idaho Falls, ID 83415-3570, USA

<sup>2</sup>Georgia Institute of Technology  
813 Ferst Drive, Atlanta, GA 30332-0405, USA

## Abstract

A computational framework for the efficient analysis and optimization of dynamic hybrid energy systems (HES) is developed. A microgrid energy system with multiple inputs and multiple outputs (MIMO) is modeled using the Modelica language in the Dymola environment. The optimization loop is implemented in MATLAB, with the FMI Toolbox serving as the interface between the computational platforms. Two characteristic optimization problems are selected to demonstrate the methodology and gain insight into the system performance. The first is an unconstrained optimization problem that optimizes intrinsic properties of the base generation, power cycle, and electrical storage components to minimize variability in the HES. The second problem takes operating and capital costs into consideration by imposing linear and nonlinear constraints on the design variables. Variability in electrical power applied to high temperature steam electrolysis is shown to be reduced by 18% in the unconstrained case and 11% in the constrained case. The preliminary optimization results obtained in this study provide an essential step towards the development of a comprehensive framework for designing HES.

*Keywords: hybrid energy systems; dynamic simulation; optimization; renewable energy; FMI*

## 1 Introduction

Hybrid energy systems (HES), which may combine multiple energy resources to achieve improved performance or cost efficiency, have attracted consider-

able attention in the United States and internationally due to developments in renewable energy technologies as well as economic, political, and environmental concerns regarding existing energy infrastructures. One notable challenge to the design and deployment of HES is the difficulty in modeling the operation and performance of such systems. Traditionally, electricity is produced from baseload generation (e.g., nuclear and coal plants), which operate at near steady state conditions with little variability. In contrast, renewable energy sources (e.g., wind turbines and concentrated solar plants) are highly dynamic with very significant variability. The addition of a renewable component to the energy infrastructure creates an enormous increase in the number of possible operational situations that must be considered, and thus greatly complicates the design process. Although it is well understood that a large energy storage device capable of achieving high rates of charge and discharge is necessary for mitigating the variability of renewable energy sources, the design and control of HES makes little use of proper design optimization methodology. In this study, we seek to apply optimization methods in order to gain a better understanding of the complex dynamics governing HES, and to obtain insights towards how to properly design such systems to maximize performance and cost efficiency.

To achieve the desired improvements in HES design, appropriate simulation and optimization tools need to be selected. The Modelica language is well suited for studying complex systems such as HES due to its object-oriented structure and acausal approach to modeling. On the other hand, the MATLAB environment provides a variety of optimization methods,

including those available in the Optimization Toolbox. In numerical optimization, it is of critical importance to automate the simulation process, because manual iterations quickly become impractical as the problem size is increased. Since our HES model and optimization algorithms are implemented in different software environments, a suitable tool for coupling diverse numerical tools is required. For this purpose, we use Functional Mockup Interface (FMI), which has been successfully demonstrated for both model exchange [1] and co-simulation [2]. In the present framework, we apply the FMI for model exchange.

The organization of the paper is as follows. First, we introduce the computational framework established in this study, with a brief summary of the HES model formulation and optimization approach. We then present two benchmark optimization problems to demonstrate the established methodology, followed by a discussion of the simulation and optimization results. Finally, we offer some concluding remarks in addition to an overview of future research directions related to this topic.

## 2 Computational framework

An illustration of the computational framework established in this paper is provided in Figure 1.

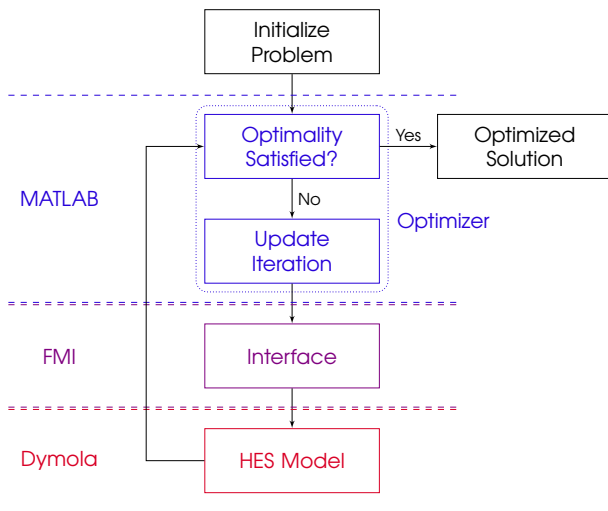


Figure 1: Schematic of computational framework.

Note that, as previously mentioned, the FMI is used to allow the MATLAB-based optimizer to modify inputs to the HES and setup additional simulations by building a Functional Mockup Unit (FMU) of the Modelica HES model. Also note that the optimization loop conducted in MATLAB may require a large num-

ber of iterations before converging to the optimized solution, and thus it is important that the HES model be sufficiently robust and flexible to handle a variety of potential operating scenarios. In the following section, we present the HES model considered in this work and discuss the concepts used in creating the model to ensure this robustness.

### 2.1 HES model

As mentioned above, the HES model considered in this work is implemented in the Dymola environment of the Modelica modeling language. As shown in Figure 2, the HES model has a configuration that includes multiple energy inputs and outputs. In this case, thermal and electrical energy in excess of the demand to the grid is dynamically distributed to a high temperature steam electrolysis (HTSE) process, which produces chemical products (hydrogen and oxygen) to complement the electricity produced by the HES. In a traditional hybrid energy system, multiple input energy sources are combined to provide a single output (MISO), i.e., electricity. The drawback of such a configuration is that in case the energy supply exceeds the demand, excess energy is either wasted or must be transferred to large storage devices, which are expensive. Therefore, the MIMO system can be expected to offer significant advantages in flexibility, utilization and efficiency over the MISO system, as reported in, for example, [3, 4].

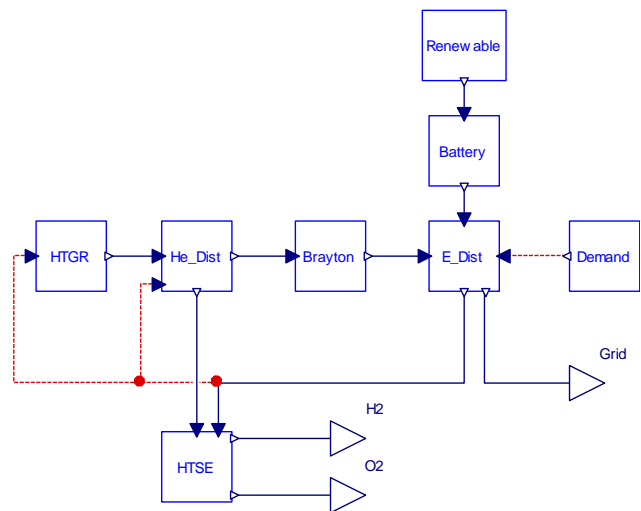


Figure 2: Model of Advanced HES configuration, implemented in Dymola.

Note that the HES considered here contains two types of energy flows: thermal (in the form of heated helium) and electrical. The conversion of thermal to



electrical energy occurs in the Brayton power plant, where heated helium (the working fluid) carries thermal energy provided by a high temperature gas reactor (HTGR). Electrical power from the Brayton plant is combined with that produced by a wind turbine (the renewable source), which is regulated by an electrical battery. Two distribution centers, one for helium and one for electricity, are used to dynamically calculate the amount of each energy type to be distributed to the HTSE to match the necessary electrical power output to the grid. Note that the HES model in Figure 2 includes two types of arrows: the solid (blue) arrows represent energy flows (helium and electricity), while the dotted (red) arrows represent information flows. For example, the amount of electricity distributed to the HTSE unit is needed at the HTGR and helium distribution center to calculate the required mass flow rate of helium. Similarly, the electricity distribution center makes use of information about the electricity demand in the grid to calculate the relative amount of electricity to be dynamically distributed between the grid and HTSE. For computational simplicity, detailed components within the subsystems such as pipes, valves, turbines, etc. are not considered in this initial effort; the system dynamics are instead captured by a series of transfer functions, switches, logic blocks, and other signal-based elements. The plan is to eventually replace these with models that more accurately capture the relevant physics; this topic is discussed further in the *Conclusions and future work* section. We next detail the models for the individual subsystems and components in the present computational framework.

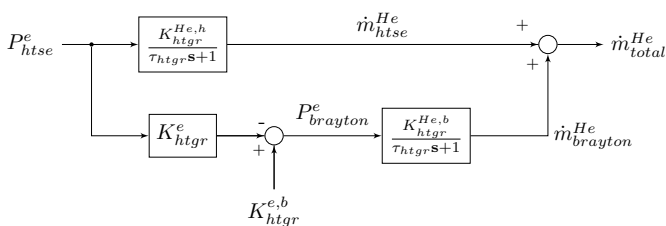


Figure 3: High temperature gas reactor model.

A schematic diagram for the HTGR is shown in Figure 3. The electrical power entering the HTSE ( $P_{htse}^e$ ) is provided by the electricity distribution center, and is the input to the HTGR. The HTGR model then uses this information to calculate the amount of high temperature helium that would correspondingly be needed by the HTSE. Also note that the total amount of helium entering the Brayton plant is controlled by the amount of electricity directed to the HTSE. In an actual reactor, the operating conditions and output should ide-

ally be constant, as fluctuations can cause excessive wear on the components and significantly shorten the lifespan of the system. Therefore, the variability in the HTGR generation will be an important component to the cost function used in the optimization of the HES in future work.

Revisiting Figure 2, we note that the mass of heated helium from the HTGR then enters the helium distribution center, where some of the helium is channeled to the HTSE for chemical production and the remainder is sent to the Brayton plant for electricity generation. As in the HTGR, the amount of electricity entering the HTSE is used as an input to the helium distribution center to calculate the flow of helium that needs to be directed to the HTSE to achieve near-stoichiometric conditions there. This is accomplished with the following equations:

$$\dot{m}_{brayton}^{He} = \dot{m}_{total}^{He} - \dot{m}_{htse}^{He} \quad (1)$$

$$\dot{m}_{htse}^{He} = K_{htse}^{He} P_{htse}^e \quad (2)$$

The Brayton plant is represented by a first-order transfer function

$$H_{brayton}(s) = \frac{K_{brayton}^e}{\tau_{brayton}s + 1} \quad (3)$$

where the gain  $K_{brayton}^e$  is a lumped quantity that accounts for efficiency and unit conversions, and the time constant  $\tau_{brayton}$  can be varied according to the design and operation of the Brayton plant.

The Brayton power plant is one source of electricity in the system; the other is a series of wind turbines, which serve as the renewable source in this HES configuration. Note that as shown in Figure 2, the wind power is not inputted directly to the electricity distribution center, but via a grid-scale battery. This is because the battery model does not model the charge and discharge behavior of the battery directly, but rather the operational impact of the battery on dampening excessive unsteadiness due to high variability in available wind power. The renewable power is modeled as a time-varying input signal to the battery based on available wind speed data from the National Renewable Energy Laboratory (NREL), for a site in Wyoming.

As shown in Figure 3, there are four operating regimes for a wind turbine, separated by critical wind speed values. At wind speeds below a minimum cut-in velocity, there is insufficient kinetic energy in the wind to cause any rotation, and thus no electrical power is produced. At wind speeds above a cut-out velocity, a braking system is activated for safety reasons,

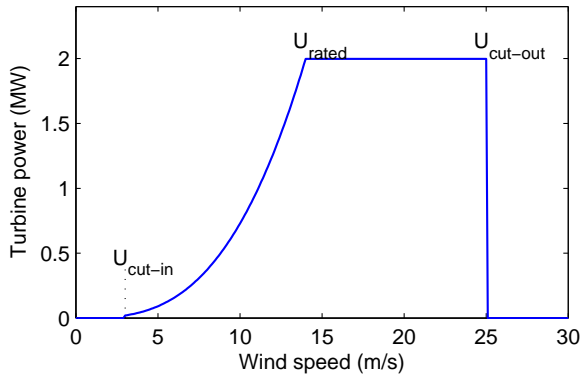


Figure 4: Turbine power vs wind speed.

and again no power is produced. Between the rated and the cut-out velocity values, the turbine provides a steady maximum power level, also known as the rated power. Finally, for the range between the cut-in and rated speeds, the power is calculated by the following equation:

$$P = \eta \frac{1}{2} \rho U^3 \frac{\pi d^2}{4} \quad (4)$$

where  $\eta$  is the conversion efficiency of the wind turbine,  $\rho$  is the density of the air at the site,  $U$  is the wind velocity, and  $d$  is the diameter of the turbine blades. In essence, this equation relates the power delivered by the turbine to the amount of kinetic energy available in the wind, via an overall lumped efficiency number. At a typical site, the majority of the turbine operation occurs in this regime.

Like the Brayton plant, the battery is modeled using a first-order transfer function:

$$H_{battery}(s) = \frac{1}{\tau_{battery}s + 1} \quad (5)$$

We assume the maximum power delivered by the battery is equal to the renewable power, so a unity gain is used. It is important to not confuse the time constant  $\tau_{battery}$  with the charge or discharge rate of the battery. Instead, the time constant is used to characterize the smoothing effect that the battery would have on the electricity delivered by a renewable and battery arrangement.

Electrical power from the Brayton plant  $P_{brayton}^e$  and the renewable and battery arrangement  $P_{battery}^e$  are combined in the electricity distribution center. The electrical distribution center contains a logic block that dynamically calculates the distribution of electricity to the grid and HTSE. When the combined electrical power  $P_{avail}^e$  is less than the grid demand, all available power is directed to the grid. When excess power is

available, it is directed to the HTSE. This is modeled using the following equations:

$$P_{avail}^e = P_{brayton}^e + P_{battery}^e \quad (6)$$

$$P_{grid}^e = \begin{cases} P_{demand}^e & \text{if } P_{avail}^e > P_{demand}^e \\ P_{avail}^e & \text{otherwise} \end{cases} \quad (7)$$

$$P_{htse}^e = \begin{cases} P_{avail}^e - P_{demand}^e & \text{if } P_{avail}^e > P_{demand}^e \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

Finally, electricity and helium energy flows are combined in the HTSE system, whose corresponding model is shown in Figure 5.

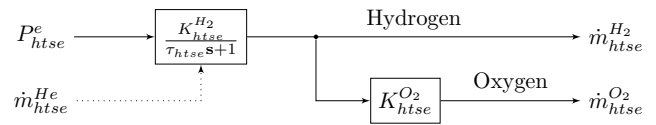


Figure 5: High temperature steam electrolysis model.

As illustrated by the dotted line in Figure 5, the mass flow rate of helium  $\dot{m}_{htse}^{He}$  enters the HTSE but is not used in any calculations. This is because the helium distribution center ensures that the appropriate flow of helium is directed to the HTSE in accordance with the amount of electricity being consumed there. Therefore, only one of the two parameters is needed in the HTSE model.

Values of all simulation parameters, including gains in the blocks shown in Figures 3 and 5, can be found in Table 1 in section 3.

## 2.2 Optimization methodology

A general optimization problem can be written as a function minimization problem, as follows:

$$\text{minimize } f(x), \quad f: \mathfrak{R}^n \rightarrow \mathfrak{R} \quad (9)$$

In general, the values of the design variables  $x$  are selected within a bounded range, while satisfying inequality constraints  $c_k$  and equality constraints  $\hat{c}_j$ :

$$\text{subject to } \begin{cases} x_{lower} \leq x \leq x_{upper} \\ c_k(x) \leq 0, \quad k = 1, \dots, m \\ \hat{c}_j(x) = 0, \quad j = 1, \dots, \hat{m} \end{cases} \quad (10)$$

Solving this general optimization problem requires an algorithm that iteratively adjusts the values of the design variables until some termination criterion regarding the values of the objective and constraint functions is met, most commonly when the Karush-Kuhn-Tucker (KKT) conditions are satisfied [5]. Many different types of optimizers have been developed and applied to a wide variety of engineering problems. These

include gradient-based and gradient-free methods, as well as hybrid approaches for mixed-integer problems [6, 7] and surrogate-based methods [8, 9]. As shown in Figure 1, in this case we adopt a black-box optimization approach, in which the optimization algorithm uses the HES model output to update each iteration of the optimization loop, but does not access any state variables within the HES model. In selecting an appropriate optimization algorithm, it is important to consider the nature of the objective function and associated design variables, and match the optimizer according to the mathematical properties of the problem. In general, gradient-based methods converge more efficiently than gradient-free methods, and are thus preferred for smooth problems [10]. However, the objective function in this study, defined in section 3 as the total variability in the electrical power delivered to the HTSE, is very noisy with a large number of local minima that are problematic for gradient-based methods. Since the optimization routine is conducted in MATLAB, we select the *fminsearch* function as the optimizer. This is an implementation of the Nelder-Mead simplex method [11] included in the MATLAB Optimization Toolbox.

The Nelder-Mead method creates a simplex with  $n + 1$  vertices in an  $n$ -dimensional design space, and iteratively manipulates the size and shape of the simplex using operations such as reflection, expansion, contraction and reduction based on the relative objective function values at the vertices. The method is gradient-free because only the values of the objective functions at the vertices are used, and thus it is suitable for noisy or discontinuous functions. The general Nelder-Mead method is valid for unconstrained optimization problems; to handle constraints, we modify the objective function  $f(x)$  by introducing a penalty function:

$$\hat{f}(x) = f(x) + p(x) \quad (11)$$

The penalty function  $p(x)$  is equal to zero in the feasible space, and gives a positive value when a constraint is violated. We then apply the optimizer to minimize the modified objective function  $\hat{f}(x)$  instead of the original function  $f(x)$ . We use a quadratic form of the penalty function that can be easily computed:

$$p(x) = \rho \sum_{i=1}^{m+\hat{m}} \max(0, c_i)^2 \quad (12)$$

where  $m$  and  $\hat{m}$  are the number of inequality and equality constraints, respectively. This penalty function is valid for both inequality and equality constraints [12]. The coefficient  $\rho$  must be large enough to force the

optimizer into the feasible space when a constraint is violated; a value of  $\rho = 10$  is found to be sufficient in this study.

The time constants of the HTGR, Brayton plant, and battery are designated as the design variables. Although the Nelder-Mead simplex method has been shown to suffer from poor convergence rate for problems involving a large number of design variables, it is suitable for the three variables considered here [13].

### 3 Problem formulation

Available wind velocity data are sampled at time intervals of 600 seconds; we conduct our simulations with the same time step size in order to avoid numerical errors that would arise from interpolating between data points. All simulations are conducted for a time period of one week ( $6.048 \times 10^5$  seconds), for a total of 1009 time steps per simulation. This time period is selected to balance the need to capture the effects of variability in the renewable energy input, with the need to conduct simulations at a feasible computational cost. As shown in Figure 6, the wind turbine experiences very fast dynamics, with numerous transient peaks and valleys observed in the velocity and power profiles within a single week. This high degree of variability confirms that the time period selected provides a representative sample of the overall long-term variability in renewable energy input experienced by the HES.

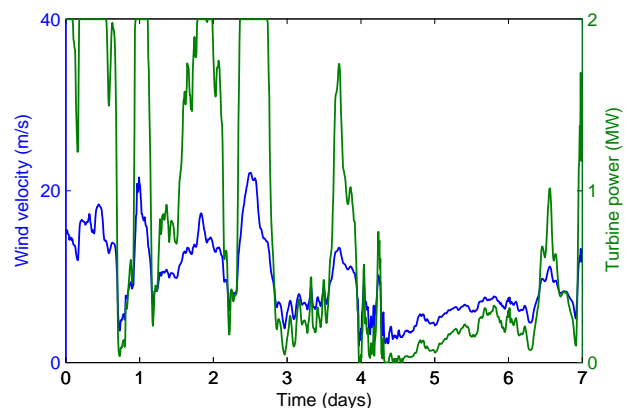


Figure 6: Wind velocity and turbine power profiles for one week sample time period.

Since the properties of the wind turbine are assumed to be constant throughout the simulations, we lump all of these values from Equation 4 together, such that each individual turbine provides maximum of 2 MW rated power. This is equivalent to an overall conversion efficiency of 55%, air density of  $1.2 \text{ kg/m}^3$  and

	Parameter	Symbol	Value	Units
HTGR	Power output	$K_{htgr}^{e,b}$	135	MWe
	He reactor gain	$K_{htgr}^{He,n}$	0.464	kg/MJ
	Power to mass flow	$K_{htgr}^{He,b}$	1.06	kg/MJ
	Electrical gain	$K_{htgr}^e$	0.0838	-
Brayton	Mass flow to power	$K_{brayton}^e$	0.9429	MJ/kg
HTSE	H <sub>2</sub> gain	$K_{htse}^{H_2}$	0.00724	kg/MJ
	O <sub>2</sub> gain	$K_{htse}^{O_2}$	7.94	-
	Time constant	$\tau_{htse}$	1	s
Wind	No. of turbines	$n_T$	15	-
	Cut-in speed	$U_{cut-in}$	3	m/s
	Rated power speed	$U_{rated}$	14	m/s
	Cut-out speed	$U_{cut-out}$	25	m/s
Grid	Target load	$P_{demand}$	100	MWe

Table 1: Fixed system properties and simulation parameters.

turbine diameter of 53 m, for an overall gain value of  $k_{wind} = 7.28 \times 10^{-4} \text{ MW}\cdot\text{s}^3/\text{m}^3$  between the cut-in and rated power wind speeds:

$$P = k_{wind} U^3 \quad (13)$$

We also simplify the scaling problem by assuming that the individual turbines, which constitute the renewable component of the HES, are separated sufficiently far apart so as to not experience mutual interference. A simple linear scaling in renewable power with the number of turbines is thus used.

The fixed simulation parameter values are listed in Table 1. The thermal and electrical outputs from all system components are initialized to be zero in order to analyze the impact of the initial start-up phase on the overall system performance, and a constant electricity profile of 100 MWe delivered from the HES to the grid is assumed. For simplicity, we select a fixed HTGR power output (135 MWe) that exceeds this output level to ensure that the HTSE unit can be operated continuously even when no renewable power is provided to the system. This is necessary because frequently shutting down the HTSE process entails a high cost while also making the system susceptible to damage.

Given these considerations, it is logical to design HES in which energy flow variability is minimized. For this purpose, we define the objective function to be minimized in our optimization problem as the total amount of variability in the electrical power input to the HTSE:

$$P_{var}^e(t) = |P_{htse}^e(t) - \bar{P}_{htse}^e| \quad (14)$$

$$f(x) = \int_0^{t_f} P_{var}^e(t) dt \quad (15)$$

where the quantity  $\bar{P}_{htse}^e$  is the time-averaged value of electrical power input to the HTSE.

For optimization, this integral can be approximated by first computing  $\bar{P}_{htse}^e$  and then applying a numerical integration scheme to the HES simulation data. Using the trapezoidal method, we obtain the following form of the objective function used by the optimizer:

$$f(x) = \sum_{i=1}^n (t_i - t_{i-1}) \frac{P_{var,i}^e + P_{var,i-1}^e}{2} \quad (16)$$

where  $n = 1009$  is the number of time steps.

We consider two optimization problems, one unconstrained and one constrained. Three design variables are considered in both cases: the time constants for the battery ( $\tau_{battery}$ ), HTGR ( $\tau_{htgr}$ ), and Brayton cycle ( $\tau_{brayton}$ ). In the unconstrained case, the design variables are bounded but allowed to vary independently. An arbitrary baseline case ( $\tau_{battery} = 3600$  s,  $\tau_{htgr} = 1200$  s,  $\tau_{brayton} = 600$  s) is used as the starting point for the optimization routine, and the following upper and lower bounds shown in Table 2 are specified.

Variable	Minimum	Maximum
$\tau_{battery}$ (s)	1800	18000
$\tau_{htgr}$ (s)	240	3600
$\tau_{brayton}$ (s)	180	3600

Table 2: Upper and lower bounds on design variables.

From an understanding of the physics governing the performance of the system, we can expect the optimizer to converge towards these bounds in the absence of constraints. This is because a larger time constant for the battery allows greater smoothing of the variability in the wind power profile, thus reducing the variability of electricity distributed to the HTSE. Conversely, a smaller time constant for the HTGR and power cycle allows them to track the electrical power demanded by the grid more closely, which again would reduce variability in electricity directed to the HTSE. Due to this understanding of the expected solution, the unconstrained optimization problem serves as a good benchmark case to verify whether the optimizer performs as expected.

Of course, the unconstrained case provides limited insight into the design of HES, because there are important constraints that need to be considered. As the time constant of the battery increases, so does the minimum battery size required to accommodate the necessary charging and discharging. This is illustrated in Figure 7, where the renewable power profiles, with and without the operation of the battery, are plotted together for three different battery time constants. To

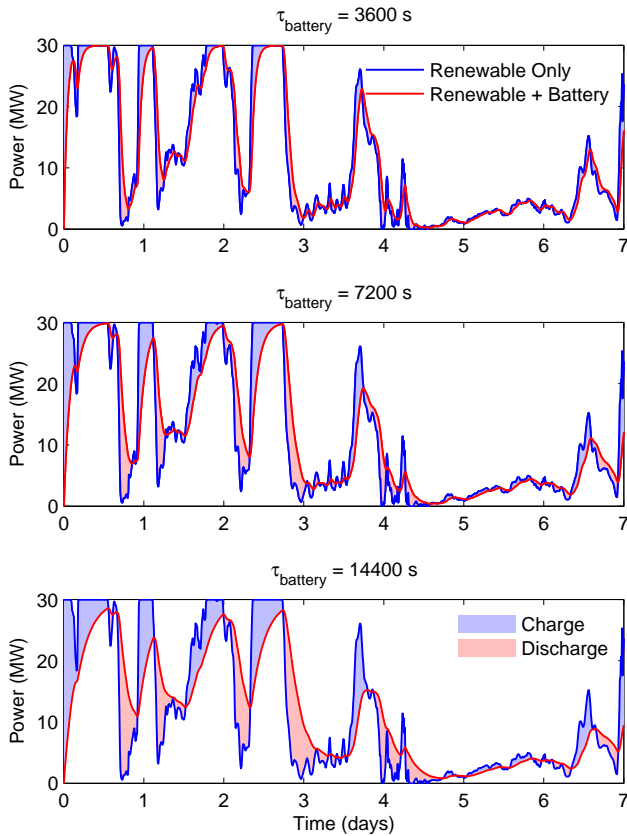


Figure 7: Effect of time constant on required battery size and delivered electrical power.

facilitate a comparison between the three cases, the difference between the two profiles is shaded (blue for when the battery is being charged, red for when it is being discharged). The required battery size (in terms of energy storage capacity) is represented by the largest shaded area, and is observed to increase with the battery time constant. Since the capital and operating costs of the battery are directly related to its size, it is clear that a constraint is needed to assess a penalty to increasing its time constant. Similarly, the time constant for the HTGR and Brayton plant cannot be arbitrarily decreased, as faster dynamics put a greater strain on the system components and lead to damage and reduced reliability - ultimately leading to increased total cost.

To address these cost issues, we consider a second optimization problem in which the following linear constraint is applied to the design variables:

$$c_1(x) = k_1 \tau_{battery} - k_2 \tau_{htgr} - k_3 \tau_{brayton} - k_4 \leq 0 \quad (17)$$

where the weighting coefficients ( $k_1=1$ ,  $k_2=3.5$ ,  $k_3=1$ ) reflect the relative cost associated with each component. The nuclear reactor is the most capital-intensive of the three, and thus has the highest weight applied to

it in the constraint function. Note that the signs of the coefficients manipulate the optimizer into decreasing  $\tau_{battery}$  while increasing  $\tau_{htgr}$  and  $\tau_{brayton}$ . The coefficients are selected such that the constant term vanishes ( $k_4 = 0$ ).

In addition to the linear constraint in Equation 17, we also apply two nonlinear constraints governing the relative speeds of the HTGR and Brayton cycle:

$$c_2(x) = \frac{\tau_{htgr}}{\tau_{brayton}} - 2 \leq 0 \quad (18)$$

$$c_3(x) = \frac{\tau_{brayton}}{\tau_{htgr}} - 2 \leq 0 \quad (19)$$

These two constraints are included to ensure that neither component becomes a significant bottleneck to the other. Note that all constraints in this case are inequality constraints; our optimization problem does not consider equality constraints (i.e.,  $\hat{c}_j(x)$  in Equation 10).

## 4 Results and discussion

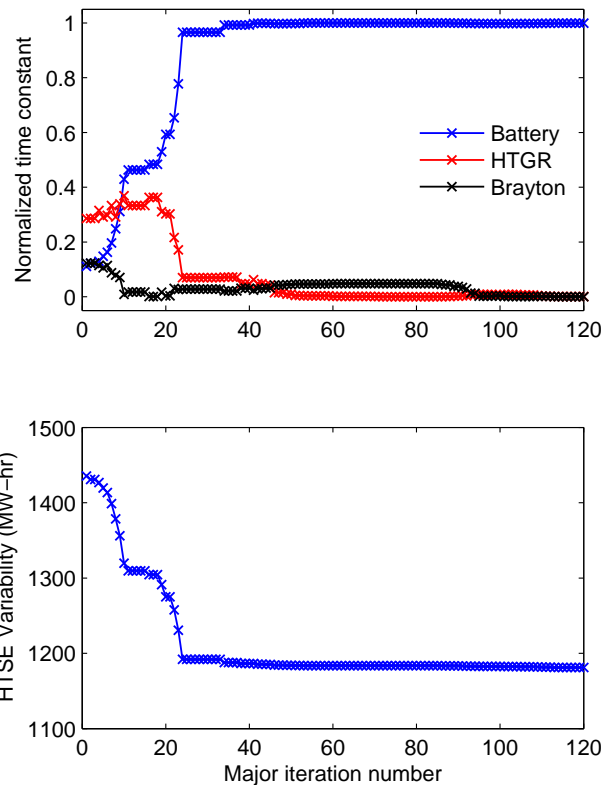


Figure 8: Progression of design variables (top) and objective function (bottom) for unconstrained optimization problem.

Optimization results for the unconstrained problem are shown in Figure 8. For illustration purposes, non-dimensional time constants are plotted by normalizing

using the upper and lower bounds reported in Table 2:

$$\tau_i^* = \frac{\tau_i - \tau_{i,\min}}{\tau_{i,\max} - \tau_{i,\min}} \quad (20)$$

As expected, the optimizer converges the battery time constant to the upper bound, and the others to their lower bounds. The value of the objective function at the optimum is  $f(x) = 1181.1$  MW-hr, which represents a reduction of 17.7% in the HTSE electrical power variability. Although this may not seem like an especially large improvement, the impact can in fact be quite substantial due to the highly capital-intensive nature of HES. A substitution of the unconstrained optimum into Equation 17 shows that the linear cost constraint is not satisfied, and thus the unconstrained optimum is not a feasible solution for the constrained problem.

As shown in Figure 9, the constrained optimum is located at  $\tau_{battery} = 15311$  s,  $\tau_{htgr} = 3600$  s,  $\tau_{brayton} = 2712$  s, with HTSE variability of  $f(x) = 1275.5$  MW-hr - a 11.2% improvement over the baseline case. The HTGR time constant experiences the most significant change compared to the unconstrained case, increasing from its minimum to its maximum bound. This is due to the high weighting assigned to the cost constraint  $c_1$ . Although the HTSE variability is increased by 6.5% compared to the unconstrained case, this loss is outweighed by the benefits in operational costs due to fast reactor dynamics. Note that in the final solution, the linear constraint  $c_1$  is active since the constraint function value is equal to numerical zero, while the nonlinear constraints  $c_2$  and  $c_3$  are inactive. This can be verified by inspection using Equations 17-19. A comparison of the optimization solutions with respect to the baseline case is summarized in Table 3.

	Baseline	Unconstrained	Constrained
$\tau_{battery}$ (s)	3600	18000	15311
$\tau_{htgr}$ (s)	1200	240	3600
$\tau_{brayton}$ (s)	600	180	2712
$f(x)$ (MW-hr)	1435.4	1181.1	1275.1

Table 3: Design variables and objective function results for baseline, unconstrained, and constrained cases.

In addition to considering the values of the design variables, constraints and objective function at the optima, it is also important to examine the HES simulations themselves to gain a better understanding of the physics occurring within the system. Figure 10 plots the electrical power profiles in the HTSE for the three cases, as well as the mean power level in each case.

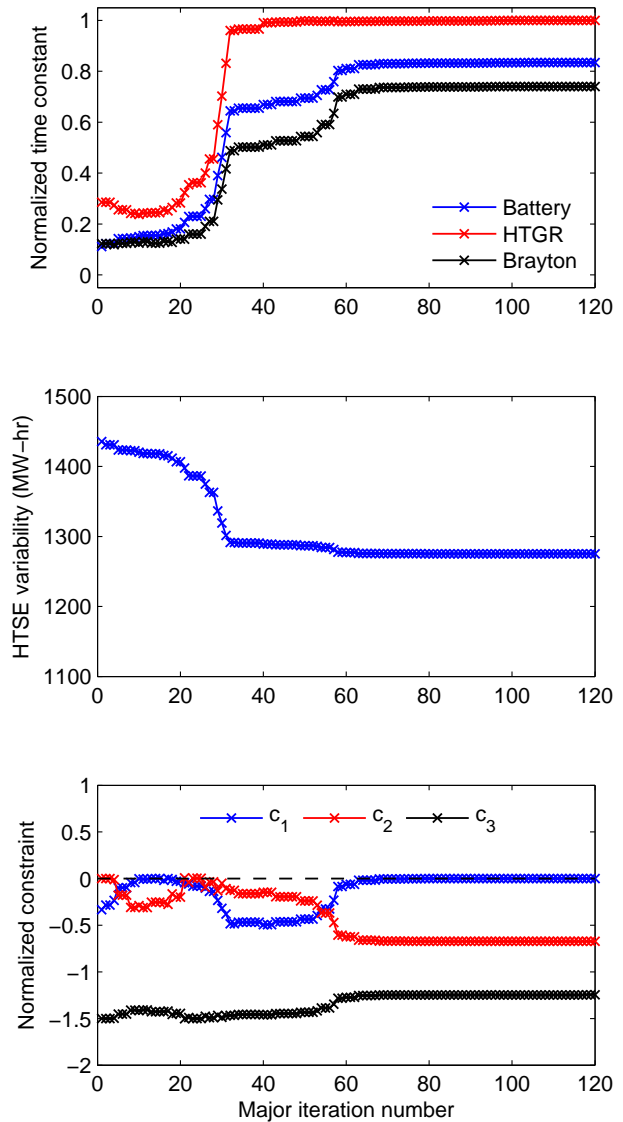


Figure 9: Progression of design variables (top), objective function (middle), and constraints (bottom) for constrained optimization problem.

The shaded area between the two is the numerically integrated area used to compute the value of the objective function. We note that in both optimization cases, not only is the total shaded area reduced, but the power profile is also generally smoother, with fluctuations of lesser amplitude and frequency. The main difference between the unconstrained and constrained solutions occurs in the initial ramping of the HTSE, where the constrained case incurs a substantial penalty due to slower reactor and power cycle dynamics. This is a direct consequence of the initial conditions in the problem formulation. For example, a long-term analysis that ignores the start-up phase of the system would likely exhibit a smaller difference between the two solutions.

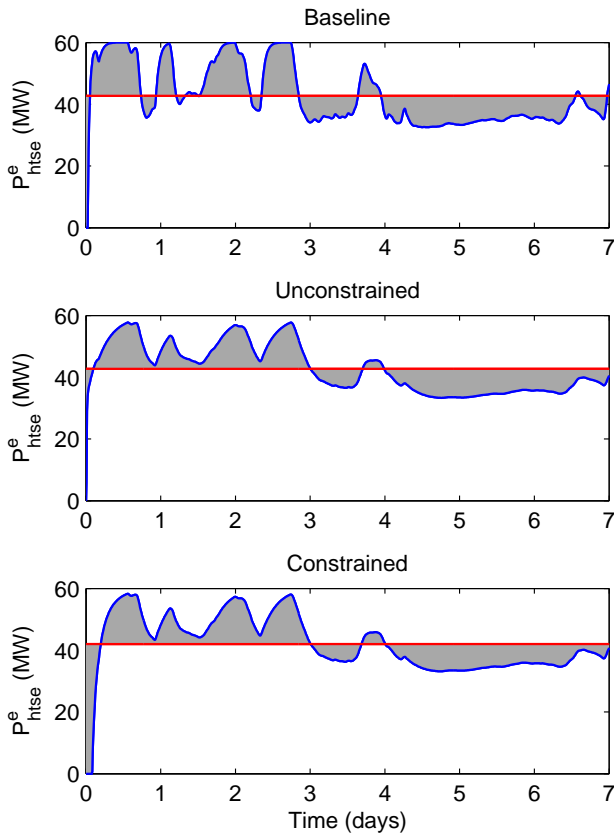


Figure 10: HTSE power profile for baseline, unconstrained, and constrained cases.

## 5 Conclusions and future work

In this paper, we have successfully established a computational framework for simulating and optimizing the performance of HES by interfacing the Modelica system model with analysis and optimization tools via the FMI for model exchange. Results from two optimization problems, one unconstrained and one constrained, illustrate the complex interplay between HES system components in the presence of high variability in the renewable power source. These preliminary results also highlight the difficulty of designing resilient and robust systems, again due to the high degree of variability and uncertainty over the lifespan of the system. The results obtained in this study represents an important step towards our goal of developing a flexible, efficient, and expandable framework for optimizing HES and achieving a level of understanding of the system dynamics and performance that would enable the deployment of HES in real world applications.

An important focus of future research efforts will be the models of the HES components. Since the main objective of the present study is to demonstrate the framework and to gain a better understanding of

the performance of HES through simulations, the simplistic but computationally efficient signal-based HES model is considered adequate in this case. However, to properly utilize the available analysis and optimization tools, a system that more closely models the relevant physical phenomena is needed. This would entail replacing the signal block component models with, for example, process models for the HTSE [14] and equivalent-circuit models for the battery [15]. To retain reasonable computational cost, reduced-order models constructed using surrogate-based techniques may also be applied [16, 17, 18]. Furthermore, a comprehensive design process for such systems must take into account a large number of variables and constraints, which cannot be investigated to a sufficient level of detail using the present model. Therefore, current efforts are aimed at creating higher fidelity, physics-based models to incorporate into the computational framework established in this study.

Another topic that warrants future investigation is the comparison of different HES configurations and assessment of new technologies. A valuable benefit of conducting the simulations in Modelica is the flexibility of the resulting framework, allowing for certain model components to be readily replaced with those representing different technologies - and for the performance of the different systems to be directly compared. For instance, we may be interested in examining the relative performance of nuclear and natural gas power plants, or the relative cost of electrical versus thermal storage in grid-scale applications.

Finally, our present efforts have focused on design optimization - i.e., optimizing certain properties of the system components. Another equally important consideration, however, is the operation of the HES after it has been properly designed. In the present study, parameters related to the system operation (such as those located in the helium and electricity distribution centers) are assigned fixed empirical values; the proper operation of the system will require incorporation of additional factors such as economics and energy efficiency into optimization problems of much greater scale.

## 6 Acknowledgment

The present research efforts have been supported by the Energy Security Initiative (ESI) at Idaho National Laboratory (INL) under the United States Department of Energy contract DE-AC07-05ID14517.

## References

- [1] T. Blochwitz, M. Otter, M. Arnold, et al. The functional mockup interface for tool independent exchange of simulation models. In *Modelica'2011 Conference, March*, pages 20–22, 2011.
- [2] M. Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *Journal of Building Performance Simulation*, 4(3):185–203, 2011.
- [3] H.E. Garcia, A. Mohanty, W.C. Lin, and R.S. Cherry. Dynamic analysis of hybrid energy systems under flexible operation and variable renewable generation—part i: Dynamic performance analysis. *Energy*, 52:1–16, 2013.
- [4] H.E. Garcia, A. Mohanty, W.C. Lin, and R.S. Cherry. Dynamic analysis of hybrid energy systems under flexible operation and variable renewable generation—part ii: Dynamic cost analysis. *Energy*, 52:17–26, 2013.
- [5] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, volume 5. California, 1951.
- [6] T. Victoire and A.E. Jeyakumar. Hybrid pso–sqp for economic dispatch with valve-point effect. *Electric Power Systems Research*, 71(1):51–59, 2004.
- [7] N. Xue, W. Du, T.A. Greszler, W. Shyy, and J.R.R.A. Martins. Design of a lithium-ion battery pack for phev using a hybrid optimization method. *Applied Energy*, 158:591–602, 2014.
- [8] R.A. Moore, D.A. Romero, and C.J.J. Paredis. A rational design approach to gaussian process modeling for variable fidelity models. In *Proceedings of ASME International Design Engineering Technical Conferences*, 2011.
- [9] W. Du, N. Xue, A. Gupta, A.M. Sastry, J.R.R.A. Martins, and W. Shyy. Optimization of  $\text{LiMn}_2\text{O}_4$  electrode properties in a gradient-and surrogate-based framework. *Acta Mechanica Sinica*, 29(3):335–347, 2013.
- [10] N. Xue, W. Du, A. Gupta, W. Shyy, A.M. Sastry, and J.R.R.A. Martins. Optimization of a single lithium-ion battery cell with a gradient-based algorithm. *Journal of The Electrochemical Society*, 160(8):A1071–A1078, 2013.
- [11] J.A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.
- [12] J. Nocedal and S.J. Wright. *Penalty and Augmented Lagrangian Methods*. Springer, 2006.
- [13] J.C. Lagarias, J.A. Reeds, M.H. Wright, and P.E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.
- [14] J.E. O’Brien, M.G. McKellar, E.A. Harvego, and C.M. Stoots. High-temperature electrolysis for large-scale hydrogen and syngas production from nuclear energy—summary of system simulation and economic analyses. *International journal of hydrogen energy*, 35(10):4808–4819, 2010.
- [15] X. Hu, S. Li, and H. Peng. A comparative study of equivalent circuit models for li-ion batteries. *Journal of Power Sources*, 198:359–367, 2012.
- [16] W. Du, A. Gupta, X. Zhang, A.M. Sastry, and W. Shyy. Effect of cycling rate, particle size and transport properties on lithium-ion cathode performance. *International Journal of Heat and Mass Transfer*, 53(17):3552–3561, 2010.
- [17] A. Gupta, J.H. Seo, X. Zhang, W. Du, A.M. Sastry, and W. Shyy. Effective transport properties of  $\text{LiMn}_2\text{O}_4$  electrode via particle-scale modeling. *Journal of The Electrochemical Society*, 158(5):A487–A497, 2011.
- [18] W. Du, N. Xue, A.M. Sastry, J.R.R.A. Martins, and W. Shyy. Energy density comparison of li-ion cathode materials using dimensional analysis. *Journal of The Electrochemical Society*, 160(8):A1187–A1193, 2013.



# Industrial application of optimization with Modelica and Optimica using intelligent Python scripting

K. Dietl<sup>a</sup>, S. Gallardo Yances<sup>a</sup>, A. Johnsson<sup>b</sup>  
J. Åkesson<sup>c</sup>, K. Link<sup>a</sup>, S. Velut<sup>c</sup>

<sup>a</sup>Siemens AG, Energy Sector, Erlangen, Germany

<sup>b</sup>Lund University, Department of Automatic Control, Lund, Sweden.

<sup>c</sup>Modelon AB, Lund, Sweden.

## Abstract

This paper shows how different kinds of optimization related task such as offline optimization or optimal control are solved using a combination of Modelica, Optimica, JModelica.org and Python. The application examples presented in this paper are all real industrial applications in the field of Combined Cycle Power Plants. Therefore different workflows have to be combined to solve the underlying task. This paper shows that these workflows can be conveniently connected using Python.

*Keywords: Dynamic optimization, Nonlinear Model Predictive Control, Extended Kalman Filter*

## 1 Introduction

Using simulation models to study plant behavior is state of the art today. So more and more attention is paid to applications related to optimization tasks. This includes e.g. offline optimization of plants, optimal plant control or parameter estimation using measurement data. These tasks often need different parts as initialization, simulation and optimization.

This paper shows a methodology which combines the optimization platform JModelica.org [1], the modeling language Modelica, an optimization extension to Modelica (Optimica) and a scripting environment (Python [2][1]) in order to solve the different optimization tasks mentioned above.

Each optimization task is illustrated by an industrial application.

The paper is structured as follows: Section 2 gives some background information about Optimica and Python, while section 3 explains the different industrial applications with focus on scripting. Section 4 summarizes the results of the paper.

## 2 Background

JModelica.org is an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems. A unique feature of JModelica.org is the support for the extension Optimica. Optimica enables users to conveniently formulate optimization problems based on Modelica models using simple but powerful constructs for encoding of optimization interval, cost function and constraints. For user interaction JModelica.org relies on the Python language. Python offers an interactive environment suitable. The following subsections give an introduction to Optimica and show the advantages of the scripting language Python.

### 2.1 Optimica

The Optimica extension mainly consists of an additional class, *optimization*, which includes the attributes such as *startTime*, *stopTime* and *objective*, which specify the objective function of the optimization problem. Another supplement is the *constraint* section, which can handle different kinds of linear and non-linear equality- and inequality constraints.

A wide range of optimization problems may be solved formulated with Optimica, for example: parameter estimation, tracking, optimal control and so on. Here is a simple example, an optimization problem, based on the double integrator:

$$\min_{u(t)} \int_0^{t_f} 1 dt$$

Subject to:

$$\begin{aligned} \dot{x}(t) &= v(t) & x(0) &= 0 \\ \dot{v}(t) &= u(t) & v(0) &= 0 \\ x(t_f) &= 1 & v(t_f) &= 0 \\ v(t) &\leq 0.5 & -1 &\leq u(t) \leq 1 \end{aligned}$$

For this problem,  $t_f$  is considered to be free and the objective is to minimize the time it takes to transfer the states from (0, 0) to (0, 1) without violating the constraints. The corresponding problem formulated in Modelica and Optimica looks like this:

```

model DoubleIntegrator
  Real x(start=0);
  Real v(start=0);
  input Real u;
equation
  der(x)=v;
  der(v)=u;
end DoubleIntegrator;

optimization DIMinTime (
  objective= finalTime,
  startTime=0,
  finalTime(free=true, initialGuess=1))

  extends DoubleIntegrator(
    u(free=true, initialGuess = 0.0));
constraint
  x(finalTime)=1;
  v(finalTime)=0;
  v<=0.5;
  u>=-1;
  u<=1;
end DIMinTime;

```

The attribute *free* = true indicates that the variable is an optimization variable and the attribute *initialGuess* provides the numerical solver with an initial guess.

The optimization problem is solved numerically and there is hence an aspect of discretization to consider, but this is considered outside of Optimica. Optimica only represents the mathematical formulation of the problem and several different solver algorithms can be used for solving the different problems at hand. For details on Optimica, see [9].

## 2.2 Python

Python is open source and a very powerful dynamic programming language that is used in a wide range of application domains. Especially interesting features of Python related to the applications covered in this paper are how well it works with other tools and its scripting possibilities. There are for example interfaces for Gnuplot [10] and Matplotlib [11], which are both suitable for plotting purpose.

Python is compatible with Optimica and it is possible to interact with Modelica and Optimica models through Python scripting (the Python interface of the Jmodelica.org platform is great for this). For example, it enables the possibility to do parameter manipulations and to perform simulation and optimization for a variety of setups. The discretization and solver options (e.g. tolerances) for simulation and optimization can easily be set through the Python script.

The Modelica and Optimica models consists of a set of states, algebraic variables and input variables and they are represented by two model object types; FMU (Functional Mock-up Unit) and JMU (JModelica Model Unit), respectively. FMUs are mainly used for simulation and they follow the FMI-standard (Functional Mock-up Interface), which specifies how the models should be represented and stored [16]. JMUs are mainly used for optimization and follow the JModelica.org standard, similar to the FMI standard for FMUs. For more details see [14]. Both model object types can be imported in to Python. Besides JModelica.org the open source simulation and optimization tool OpenModelica [8] supports Python with its interface called OMPython enabling the user to use the modeling and simulation capabilities of OpenModelica within the Python environment.

The Python packages SciPy [12] and NumPy [13] support linear algebra and matrix operations and are useful when scripting, both for pre- and post-processing, in plotting and in the implementation of algorithms.

Below is a simple example that demonstrates what scripting in Python may look like, using JModelica.org. The script solves the optimization problem described in section 2.1. DoubleIntegrator.mo contains the model and DIMinTime.mop contains the optimization model. A JMU object is created based on these.

```

# Importing necessary packages and functions
import numpy as N
from pymodelica import compile_jmu
from pyjmi import JMUModel
import matplotlib.pyplot as plt

# Compiling a JMU- object for optimization
# based on the double integrator
jmu_name= compile_jmu("DIMinTime",
    ["DoubleIntegrator.mo", "DIMinTime.mop"])

# Loading the JMU-object
model_opt = JMUModel(jmu_name)

# Calling the optimization function with
# default settings
res = model_opt.optimize();

# Plotting the results
x= res['x']
v =res['v']

# u is the optimal trajectory
u =res['u']
time = res['time']

plt.plot(time, x, ':k', time, v, '--k',
    time, u, '-k')
plt.grid(True)

# Increasing the plot window to show results
plt.ylim([-1.5,2])
plt.legend(('x','v','u'))
plt.title('Simple example')
plt.show()
    
```

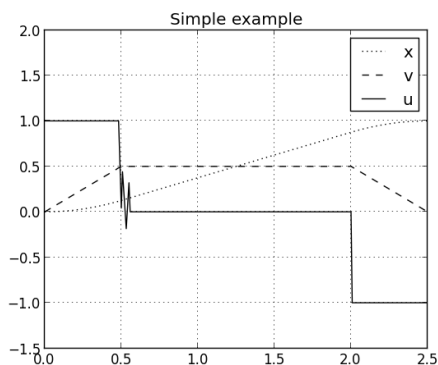


Figure 1: The optimization results

### 3 Case studies

The JModelica.org platform can be used to perform offline optimization. Python scripting is used for pre- and post-processing.

#### 3.1 Offline optimization: Start-up of a combined cycle power plant

The aim of this offline optimization is to maximize power output of gas and steam turbines without violating given constraints on temperature differences in

several components. The optimizer shall find optimized control inputs for gas turbine power and four different control valves (see Figure 2).

This model is an extension of the model presented in [6]: the model scope has been enlarged to also include high pressure (HP) and intermediate pressure (IP) turbine and the corresponding valves.

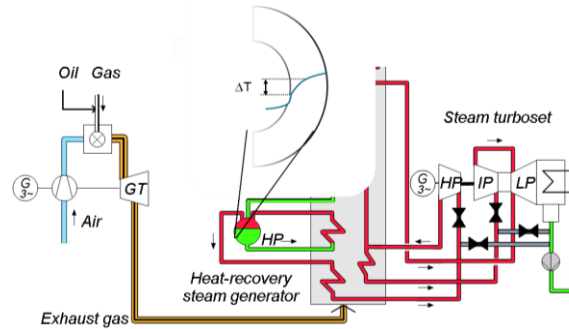


Figure 2: Model for startup optimization

##### 3.1.1. Python Scripting

Figure 3 illustrates the steps written in Python to perform offline optimization.

Since the gas turbine power is fixed until synchronization of the gas turbine, the optimization can not start at  $t = 0s$ , but only after synchronization, at  $t = t_{opt\_start}$ . Therefore a first simulation determines the initial conditions for the optimization. Additionally to the state initialization at optimization starting point, the optimization algorithm also needs a guess trajectory for all variables. If no explicit guess trajectory is supplied it can be obtained using a second simulation. After the optimized control input has been obtained, several post-processing steps are taken (see Figure 3). In this specific application, the optimization result and the result of the first simulation (until synchronization of the gas turbine) are concatenated to obtain the complete time-dependent behavior of the plant. Also it proved to be useful to run a check, whether all variables are well scaled and whether the constraints are also kept between the collocation points. It has to be stated, that the Python scripting environment offers a very convenient and flexible way to include different pre- and post-optimization tasks.

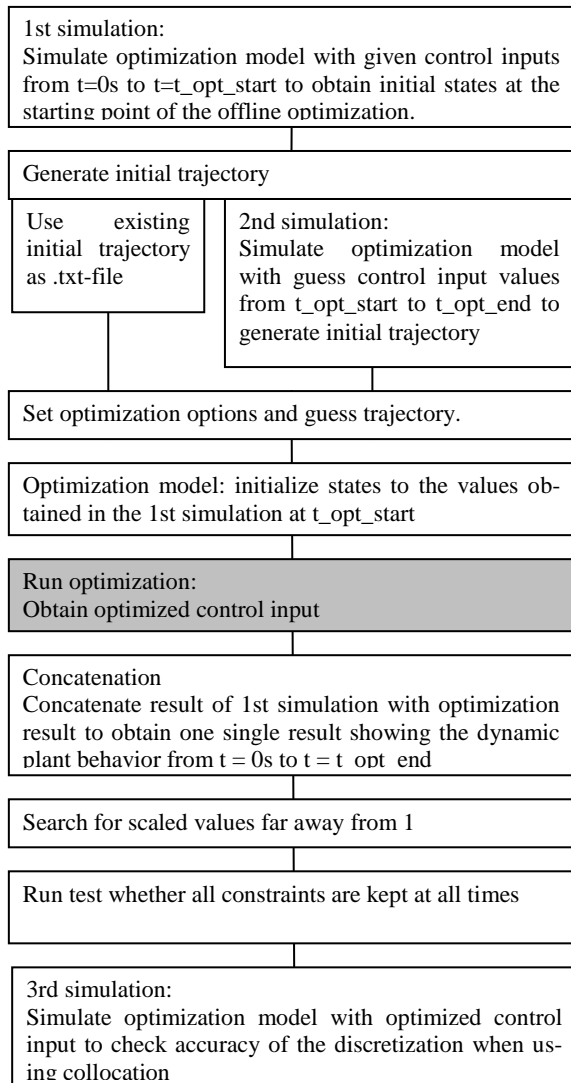


Figure 3: Steps written in Python when performing offline optimization

### 3.1.2. Results of the offline startup optimization

An offline startup optimization of the system shown in Figure 2 has been performed where the characteristics are given below:

Constraints:

- Model equations
- Maximum pressure of 170 bar
- Pressure dependent maximum wall temperature difference for several components (i.e. HP and IP turbine casing, heat exchanger manifolds etc.)
- Minimum mass flow rate change for a certain mass flow range
- Maximum negative GT power derivative

Control inputs:

- Gas turbine power
- HP and IP turbine bypass valve
- HP and IP turbine control valve

Optimization objective:

- Maximization of total power output (gas turbine power + power of HP and IP turbine)

Some optimization results are presented in the following figures.

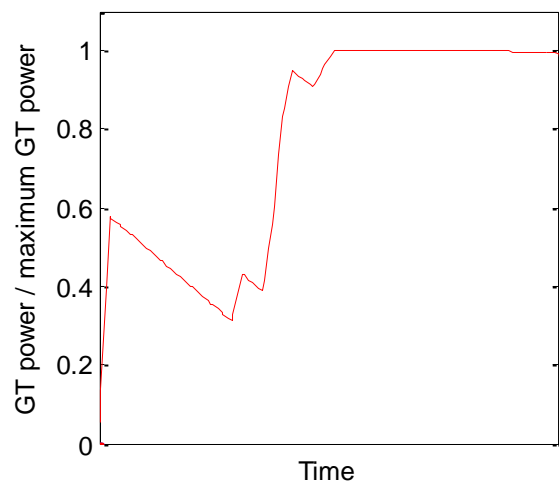


Figure 4: Normalized gas turbine (GT) load (actual divided by maximum gas turbine load).

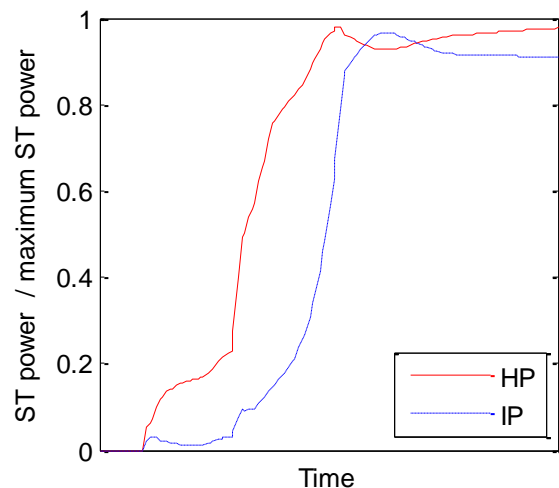
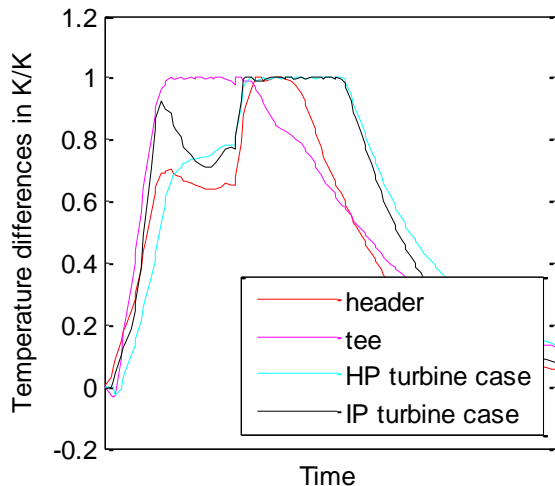


Figure 5: Normalized steam turbine power (solid line: HP turbine, dashed line: IP turbine)



**Figure 6: Normalized temperature differences (actual temperature difference divided by maximum allowable temperature difference) in critical components**

It can be seen that the optimizer first strongly increases the gas turbine power to maximize the power output (Figure 4). Then the gas turbine power is decreased again to keep the temperature difference in the critical components below their maximum value (Figure 6).

### 3.2 Nonlinear Model Predictive Control with State Estimation: Extended Kalman Filter

The basic concept of Nonlinear Model Predictive Control (NMPC) is to use a dynamic model to forecast system behavior and optimize the forecast to produce the best decision. In practice an optimal control problem is solved over a finite future horizon, but only the first optimal control signal is applied to the system. Then the optimization horizon is shifted and the calculations are repeated.

The solution of the optimal control problem depends on the initial state of the model which is the current state of the plant. In general, measurements are disturbed by noise or are missing, so that a state estimation algorithm is needed to determine the initial states under consideration of the past record of measurements.

#### 3.2.1 Python Scripting

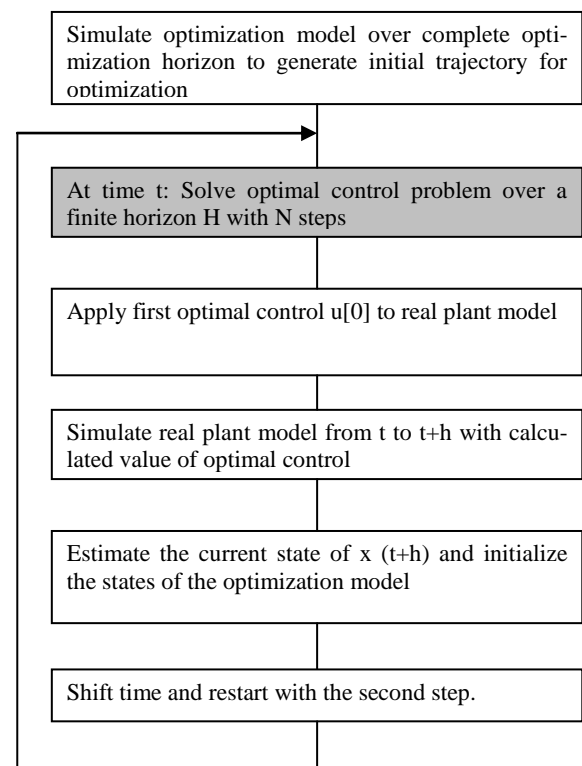
The python script for optimization with JModelica.org looks as in Figure 7.

This scheme describes the NMPC loop with two dynamic models – one model called optimization model, the other real plant model. The real plant model illustrates the real plant behavior and is more detailed than the optimization model. In future the

real plant model will be replaced by measurements of the real plant.

The NMPC loop starts with the generation of an initial trajectory for the optimization. As an alternative the optimization can be initialized with an optimization result too. The initial trajectory is generated by simulating a FMU of the optimization model.

The second step is to solve the optimization problem from  $t$  to  $t+H$ , where  $t$  is the actual time and  $H$  the length of the finite optimization horizon (see Figure 8). The optimization horizon is divided into  $N$  steps, but only the first control signal is applied to real plant model and the model is simulated to get the new state of the plant. The following step is the state estimation which is explained in more detail below. After the initial state of the model is updated the optimization horizon is moved and the optimization is solved again from  $t = t+h$  ( $h = H/N$ ) to  $t+2 \cdot h$ . All steps will be repeated until the final time of the optimization  $t_{opt\_end}$  is reached.



**Figure 7: Steps written in Python performing NMPC with state estimation**

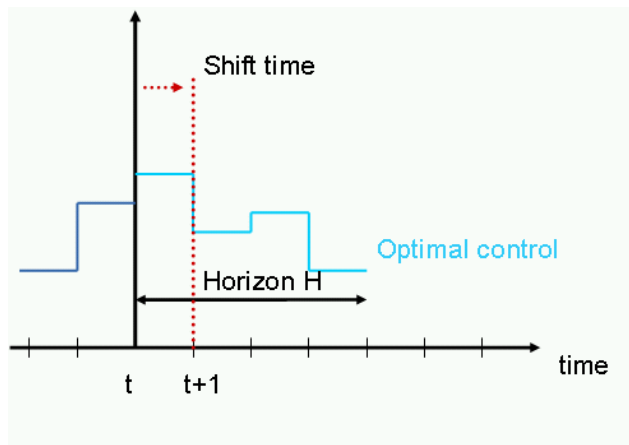


Figure 8: NMPC scheme

The strategy of Extended Kalman Filter (EKF) is used for state estimation. The Kalman filter originates from probability theory and it is well established that the Kalman filter is the optimal state estimator for a linear system affected by white noise. The Kalman filter minimizes the estimation error by considering past data of the system. This can be described in a recursive way which is convenient for implementation purposes [15]. Concerning the notation,  $\hat{x}_{t+1|t}$  corresponds to the estimation of  $x$  at  $time = t+1$  given the information at  $time = t$ .

The setup of EKF is according to Figure 9. The estimation consists of two main steps; the prediction and the correction. In the prediction step, the state values at time  $t+1$  are estimated from the system representation ( $\hat{x}_{t+1|t}$ ). The covariance matrix of the estimation error at the prediction step ( $P_{t+1|t}$ ) is also updated in this step.

In the correction step, the Kalman gain ( $K_{t+1}$ ) is updated. It is then used to derive the corrected state estimation ( $\hat{x}_{t+1|t+1}$ ) by combining the result of prediction step and the latest plant measurements ( $y_{t+1}$ ). The covariance matrix for the estimation error at the correction step ( $P_{t+1|t+1}$ ) is also updated here. The steps are described in more detail below.

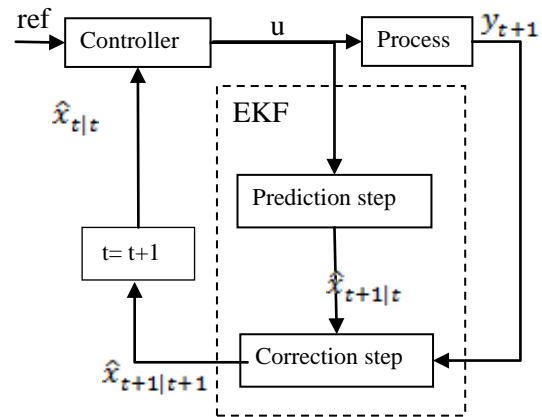


Figure 9: Structure of EKF, where  $ref$  corresponds to the control reference,  $u$  to the control signal, and  $x$  to the state estimation at different stages and  $y$  to the real plant measurements.

The EKF is an extension of the Kalman filter for nonlinear process models and the approach is basically the same as in the linear case, with an additional linearization to get approximations of  $A_{t|t}$  and  $C_{t+1|t}$  matrices (using standard notation for linear systems), which are used by the filter. The linearization step was realized with the JModelica.org library `pyjmi.linearization`.

Prediction:

- $\hat{x}_{t+1|t}$  By simulating a FMU of the optimization model.
- $A_{t|t}$  By linearizing a JMU of the optimization model (at  $\hat{x}_{t|t}$ ).
- $P_{t+1|t} = A_{t|t}P_{t|t}A_{t|t}^T + Q_t$

Correction:

- $C_{t+1|t}$  By linearizing a JMU of the optimization model (at  $\hat{x}_{t+1|t}$ )
- $K_{t+1} = P_{t+1|t}C_{t+1|t}^T(C_{t+1|t}P_{t+1|t}C_{t+1|t}^T + R_t)^{-1}$
- $y_{t+1}$  By simulating a FMU of the real plant model.
- $\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}(y_{t+1} + \hat{x}_{t+1|t})$
- $P_{t+1|t+1} = P_{t+1|t} + K_{t+1}C_{t+1|t}P_{t+1|t}$

$Q$  and  $R$  represent the covariance matrices for process and measurement noise.

For this example, we assumed that both noises are uncorrelated and conform to a normal distribution.  $Q$  and  $R$  are diagonal matrices. The entries on the diagonal of the covariance matrices  $Q$  and  $R$  are the vari-

ances of each process variable and measurement and are set to 1 in most cases. Q and R were approximated to be uncorrelated in time and the diagonal elements were set to: one over the square root of the standard deviation of the corresponding state/measurement, in most cases to 1.

The EKF does not consider constraints, and this has to be compensated for in an additional step (the feasibility correction). This is important to note since the optimization strategy is interior point optimization, and no solution will be found if the starting point is outside of the feasible region.

The strategy to considering this fact was simply to use the prediction as state estimation, without the correction step, since the prediction always will be feasible.

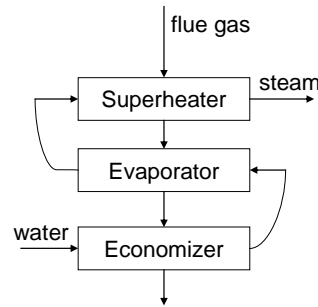
### 3.2.2 Results of NMPC with State Estimation: Extended Kalman Filter

In reality, there will be large differences between the controller model and the real plant, this is natural. However, for the evaluation of the implementation, the models were kept basically the same, with the same state representation. The optimization model was augmented in order to compensate for the differences between the plant models. The differences were approximated as constant disturbances, by introducing the additional states  $d$ , here on referred to as *the disturbance states*. These are unmeasurable states of the real plant and their values can be estimated by the EKF.

See a simple example of the augmentation below, where  $x$  represent the original states,  $f(x, u)$  represents the process with input  $u$  and  $d$  represent the disturbance states.

$$\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k) \\ d_k \end{bmatrix}$$

The NMPC loop combined with state estimation was evaluated using the example of an enthalpy controller of the heat recovery steam generator (HRSG) (consisting of economizer, evaporator and superheater, see Figure 10) of a combined cycle power plant.



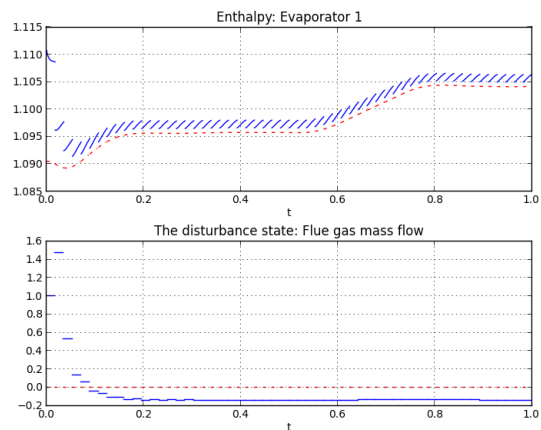
**Figure 10: Rough sketch of the system to be controlled.**

Two disturbance states were added to the optimization model (one to the flue gas mass flow rate and one to the water pressure) in order to consider differences. Additionally a parameter related to the heat transfer was modified to represent a typical modeling error in addition to initial offsets for each state.

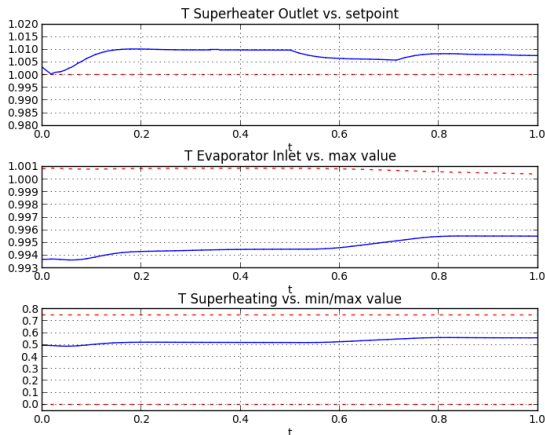
There were three objectives for the controller and they were considered in the formulation of the optimization problem:

1. Keep the steam temperature at the superheater outlet at desired set point.
2. Guarantee subcooling at the evaporator inlet by keeping the temperature below a specified maximum value.
3. Have an adequate degree of superheating at the outlet of the evaporator section.

Figure 11 and Figure 12 display the results for this setup.



**Figure 11: The progress of the state estimation, scaled according to nominal value and time. The red line represents the real plant behavior. The top plot represents one of the process states, the enthalpy at the inlet of first evaporator. The bottom plot represents the introduced disturbance state, with an initial error**



**Figure 12: The control objective, scaled according to nominal value and time. The *top plot* displays the temperature set point control; the red line represents the set point. The *middle plot* displays the sub cooling control; the red line represents the maximum value for the temperature at the economizer outlet (pressure dependent). The *bottom plot* displays the superheating control, the red lines represents the minimum/maximum degree of superheating.**

The jumping in Figure 11 is related to the correction step of the EKF. Also worth noting is that the disturbance state does not estimate the constant disturbance to 0, and this is probably due to the introduced modeling error. The disturbance state tries to compensate for this as well.

The controller performance is quite satisfying, but there are some offsets that should be considered in the future work with optimization problem formulation. The reason for the offset can for example be related to the fact that the plant did not reach steady state within the time frame for the experiment. It is not clear from Figure 11, but it is however the case. Evaluating the performance for a longer time might get rid of the offset, but this was not possible to realize at the time of the application evaluation because of some memory leaks. Modifications to the objective function could be increasing the weight on the elements in the objective function related to set point deviations.

### 3.3 Parameter estimation

Parameters are typically estimated by some form of least squares. This method minimizes the sum of the squared discrepancies between measurement and expected value.  $M$  is the number of measurements and  $N$  the number of discrete time steps.

$$\min \sum_{i=1}^N \sum_{j=1}^M \left( x_{i,j} - x_{i,j}^{meas}(t[i]) \right)^2$$

The default algorithm for solving optimal control problems and parameter estimation problems in JModelica.org is the collocation algorithm. For our application we used the Nelder-Mead method, a heuristic search method using the concept of a  $(N+1)$ -dimensional simplex, where  $N$  is the number of estimated parameters. These kinds of derivative free algorithms are implemented in JModelica.org ([2]).

The Nelder-Mead algorithm was the preferred choice, although this method is quite slow, but has the best convergence behavior especially for many measurements and a lot of parameters.

#### 3.3.1 Python Scripting

As the Nelder-Mead method is already implemented in JModelica.org as described in [9], the python script is quite simple. In a first step all measurements are imported from .mat file. Then the Nelder-Mead function `nelme` is called which solves the optimization problem and uses the defined objective function as input.

#### 3.3.2 Results of Parameter estimation

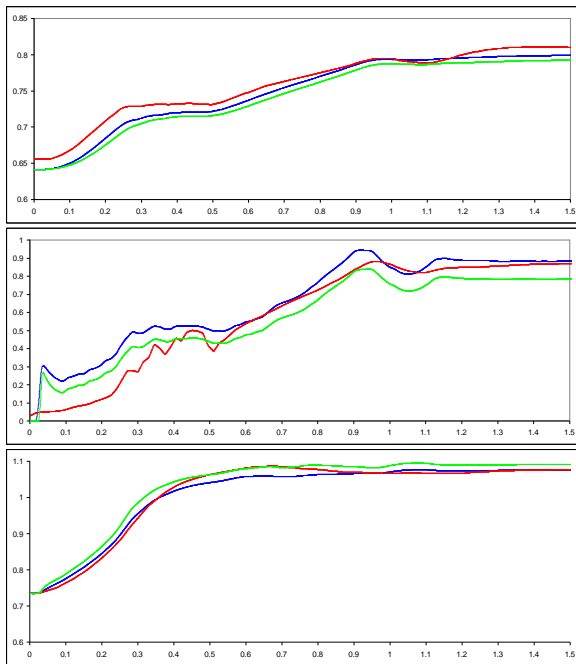
A parameter estimation of the Modelica model for optimizing the start up process of a combined cycle power plant in 3.1 has been implemented.

The real plant measurements were given for a period of 1h. The data were loaded in python as .mat file. The gas turbine power, the gas turbine mass flow, the injection mass flow and the back pressure measurement were set directly as boundaries of the model, the other measurements (wall temperatures, fluid temperatures and pressure) are used to minimize the error between the measurements and their simulated values.

The algorithm was tested for two measurement sets to verify the result, one for a hot start and one for a warm start of the power plant. As expected the estimated parameters have smaller differences, but have equal dimensions.

Figure 13 shows the simulation result with original and optimized parameters compared to measurements. As can be seen, the simulation result with the optimized parameter fits better to the measurements. Nevertheless there are still differences between the model and the real plant behavior. Uncertainty of measurements, not modeled effects and components are some reasons for the deviations.





**Figure 13: Simulation result with original parameters (green) optimized parameters (blue) and measurements (red). The top plot displays the wall temperature of the separator; the middle plot displays the reheat mass flow and the bottom plot the outlet temperature of the first superheater.**

## 4 Summary

This paper shows how different industrial optimization applications can be solved combining Modelica and Optimica with the scripting language Python.

Three different optimization tasks have been considered to improve the dynamic processes in a combined cycle power plant: offline optimization of the start-up process, online enthalpy control of the HRSG and parameter estimation for the start-up optimization.

All optimization tasks have been formulated with Optimica based on Modelica models. For pre- and post processing issues and interaction of JModelica.org and the Modelica models, the scripting tool Python was used.

As shown in section 3 ‘Case studies’, for complex optimization techniques like NMPC with combinations of several optimization and simulation steps, Python is ideally suited. For industrial applications of power plant sector the definition of such controllers inside the Modelica model is not desired and unnecessary since other interfaces (e.g. connection to database) have to be realized additionally.

The work presented in this paper is one step towards a complete online-optimization tool chain for NMPC.

## 5 Acknowledgements

The German Ministry BMBF has partially funded this work (BMBF funding code: 01IS12022A) within the ITEA2 project MODRIO [7]. Modelon’s contribution to this work was partially funded by Vinnova, through the ITEA 2 project MODRIO.

## References

- [1] JModelica.org, <http://jmodelica.org/>, viewed 2013-12-05.
- [2] Python Software Foundation. Python Programming Language - Official Website, <http://www.python.org/>, 2012, viewed 2013-12-05
- [3] A. Johnsson, Nonlinear Model Predictive Control for Combined Cycle Power Plants, Master’s Thesis, Lund University, Department of Automatic Control, 2013.
- [4] C. Andersson, S. Gedda, J. Åkesson, S. Diehl, Derivative-free Parameter Optimization of Functional Mock-up Units, 9<sup>th</sup> International Modelica Conference, Munich, Germany, 2012.
- [5] Lie, B., Haugen Finn, Scripting Modelica Using Python, Telemark University College, Porsgrunn, Norway, 2012
- [6] A. Lind, E. Sällberg, S. Velut, S. Gallardo Yances, J. Åkesson, K. Link: Start-up Optimization of a Combined Cycle Power Plant, Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany
- [7] <https://www.modrio.org/>
- [8] <https://openmodelica.org/>
- [9] J. Åkesson, Optimica—An Extension of Modelica Supporting Dynamic Optimization, 6th International Modelica Conference 2008, Modelica Association, March 2008.
- [10] GnuPlot, <http://www.gnuplot.info/>, viewed 2013-12-05.
- [11] Matplotlib, <http://matplotlib.org/>, viewed 2013-12-05.

- [12] SciPy, <http://www.scipy.org>, viewed 2013-12-05.
- [13] Numpy, <http://www.numpy.org/>, viewed 2013-12-05.
- [14] The JModelica.org User Guide, <http://www.jmodelica.org/api-docs/usersguide/JModelicaUsersGuide-1.11.0.pdf>, viewed 2013-12-05.
- [15] E. Haselting, J. Rawlings, A Critical Evaluation of Extended Kalman Filtering and Moving Horizon Estimation, <http://jbrwww.che.wisc.edu/tech-reports/twmcc-2002-03.pdf>, 2003, viewed: 2013-05-07.
- [16] The FMI-standard, <https://www.fmi-standard.org/>, viewed 2014-01-21.

# Simulation of Smart-Grid Models using Quantization-Based Integration Methods

X. Floros<sup>a1</sup> F. Bergero<sup>b1</sup> N. Ceriani<sup>c</sup> F. Casella<sup>c</sup> E. Kofman<sup>b</sup> F. E. Cellier<sup>a</sup>

<sup>a</sup>Department of Computer Science, ETH Zurich, Switzerland

<sup>b</sup>CIFASIS-CONICET, Rosario, Argentina

<sup>c</sup>Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy

## Abstract

Concepts such as smart grids, distributed generation and micro-generation of energy, market-driven as well as demand-side energy management, are becoming increasingly important and relevant as emerging trends in the design, management and control of energy systems. Appropriate modeling and design, efficient management and control strategies of such systems are currently being studied. In this line of research a very important enabling component is efficient and reliable simulation. However those energy models are typically large, stiff and exhibiting heavy discontinuities, and at the same time consist of interconnected multi-domain subsystems encompassing electrical, thermal, and thermo-fluid models. Object-Oriented (O-O) languages such as Modelica are obviously well-suited for the modeling of such systems; however, traditional state-of-the-art hybrid differential algebraic equation solvers cannot efficiently simulate these systems especially when their size grows to the order of hundreds, thousands, or even more interconnected units.

The goal of this paper is to show, through a couple of exemplary case studies, that Quantized State System (QSS) integration methods are ideally suited to solve models of such systems, as they scale up better than traditional methods with the system size, and provide time savings of several orders of magnitude, while achieving comparable numerical precision.

**Keywords:** *Quantization-Based Integration Methods, QSS, DASSL, Smart-Grids, EnergyMarket, Modelica*

<sup>1</sup>The authors contributed equally to this work.

## 1 Introduction

The growing interest in new paradigms for energy systems such as Smart Grids (SG) is posing new challenges in the control of procurement, conversion, distribution and use of energy to meet environmental and economic objectives.

Computer simulation of SG systems is a fundamental tool for production planning and control, price regulation, logistics, etc. To carry out the simulation one must deal with two problems. First, modeling complex SG systems involves taking into account components from various domains such as thermal, electrical, ventilation, etc. Each component could be developed by different specialists, possibly using different languages or formalisms that must then be coupled to produce the complete model. SG models are commonly composed of energy production facilities, energy transmission networks and usually hundreds or thousands of energy consumption units. Thus the modeling of these types of systems is a difficult task. Second, once the problem is modeled, the actual simulation of a large hybrid model (with continuous and discrete subcomponents) can turn out to be prohibitively expensive in terms of CPU time, as the scale of the system grows.

Modelica [13] is an Object-Oriented (O-O) language for modeling and simulation of complex multi-domain physical systems, described by hybrid differential-algebraic equations. In the literature several research efforts show the use of Modelica as a language for modeling SG problems [6, 7, 8, 19, 20, 21]. State-of-the-art Modelica simulation tools generate simulation code that solves the differential equations using classical numerical integration methods, such as Euler, Runge-Kutta, or DASSL, which are based on time discretization.

Another approach is the use of Quantized State System (QSS) methods [4, 14, 15] which replace time discretization by state quantization. The QSS methods have certain features (sparsity exploitation, efficient discontinuity handling, explicit stiffness treatment) that make them particularly effective for large, sparse, hybrid, dynamical systems like the SG models.

In this work we investigate the suitability of the QSS methods for simulating SG models described in Modelica and compare their efficiency, as well as simulation quality, against classical integration methods.

We shall focus on two models, first a District Cooling System taken from [5] and then an Energy Market with houses as energy consumption units adapted from [6].

The paper is organized as follows: Section 2 introduces the main concepts used along the article, then Section 3 describes the two smart-grid applications, and in Section 4 we compare different numerical integration methods. Finally, Section 5 concludes the article and outlines future work.

## 2 Background

This Section introduces the main concepts used along the remainder of the article.

### 2.1 Classical Numerical Integration Methods

The mathematical models describing Energy Management problems such as the SG are usually time dependent dynamical systems. These can be expressed in the form of a set of Differential Algebraic Equation (DAEs) or directly in a set of Ordinary Differential Equations (ODEs) as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \quad (1)$$

where  $\mathbf{x}(t)$  is the state vector.

Classical numerical integration methods discretize the time variable computing the states variables for certain time points.

We shall focus on two well-known numerical integration methods [4]:

**Runge-Kutta** An explicit variable step algorithm of fourth order.

**DASSL** An implicit variable step algorithm based on a series of Backward Difference Formulae

(BDF) of different orders of approximation accuracy.

As both methods are based on time discretization, discontinuity detection is an expensive mechanism. Also, only implicit algorithms are able to efficiently simulate stiff systems, i.e. systems that exhibit simultaneous fast and slow dynamics, without resorting to unnecessarily short time steps.

### 2.2 QSS Integration Methods

Quantized State System (QSS) methods replace the time discretization of classic numerical integration algorithms by the quantization of the state variables.

Given the ODE of Eq.(1), the first order Quantized State System method (QSS1) [16] approximates it by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), t) \quad (2)$$

Here,  $\mathbf{q}$  is the *quantized state vector*. Its entries are component-wise related with those of the state vector  $\mathbf{x}$  by the following *quantization function*:

$$q_j(t) = \begin{cases} x_j(t) & \text{if } |x_j(t) - q_j(t^-)| \geq \Delta Q_j \\ q_j(t^-) & \text{otherwise} \end{cases} \quad (3)$$

where  $\Delta Q_j$  is called *quantum* and  $q_j(t^-)$  denotes the left-sided limit of  $q_j$  at time  $t$ . The quantization function  $\mathbf{q}(t)$  in QSS methods also contains a *hysteretic* term (not shown here for simplicity) that is necessary in order to avoid illegitimate models [16].

It can be easily seen that  $q_j(t)$  follows a piecewise constant trajectory that only changes when the difference between  $q_j(t)$  and  $x_j(t)$  becomes equal to the quantum. After each change in the quantized variable, it results that  $q_j(t) = x_j(t)$ .

The QSS1 method has the following features:

- In the solution, the quantized states  $q_j(t)$  follow piecewise constant trajectories.
- The state variables  $x_j(t)$  follow piecewise linear trajectories.
- The state and quantized variables never differ by more than the quantum  $\Delta Q_j$ . This fact ensures stability and global error bound properties [4, 16].
- The fact that the state variables follow piecewise linear trajectories makes the detection of discontinuities a trivial task. Moreover, after a discontinuity is detected, its effects are no different

from those of a normal step (because changes in  $q_j$  are discontinuous). Thus, QSS1 is very efficient in simulating discontinuous systems [14].

- Each step is local to a state variable  $x_j$  (the one that reaches the quantum change), and it only provokes evaluations of the state derivatives that explicitly depend on it. This fact implies that QSS1 performs intrinsic sparsity exploitation.
- If some state variables do not change significantly, they will not provoke any step or evaluation at all. This feature reinforces the efficient sparsity exploitation.

The last two points show that QSS methods integrate each state variable at its own pace i.e. a fast changing variable would provoke more local integration steps than a slow one.

As QSS1 only performs a first order approximation, good accuracy cannot be obtained without a significant increment in the number of steps. Also as QSS1 is an explicit solver, the algorithm is not suitable for simulating stiff systems. The former limitation was solved with the introduction of higher order QSS methods like QSS2 and QSS3 [15]

For the simulation of stiff systems, a family of linearly implicit QSS methods (LIQSS) of orders 1 to 3 was also proposed in [17]. LIQSS methods are semi-implicit methods that can handle certain types of stiff systems.

In the context of this work, the efficient sparsity exploitation, the semi-implicit treatment of stiff systems and the native handling of frequent discontinuities compose the main advantages of the QSS methods.

### 2.3 Stand-Alone QSS Solver

It was shown that the behavior of the QSS approximation of Eq.(2) can be described as a Discrete Event System (DEVS)[22]. Thus, a straightforward implementation of these algorithms is through their equivalents in a DEVS simulation engine.

DEVS-based implementations of QSS methods are simple but they are not efficient. The problem is that the DEVS simulation engines waste a large amount of the computational load attending the DEVS simulation mechanism. This fact motivated the development of a stand-alone QSS solver.

Recently, the complete family of QSS methods was implemented in a *stand-alone QSS solver* coded in plain C language [9]. This solver simulates models

described in a subset of the Modelica language, called  $\mu$ -Modelica [3].

In this work all simulations are performed using the stand-alone QSS solver.

### 2.4 OpenModelica

OpenModelica [12] is an open-source Modelica-based modeling and simulation environment. OpenModelica offers different numerical integration methods for simulation, amongst them, the before mentioned DASSL and Runge-Kutta. In this article we shall use this tool as a reference to compare the performance of different integration methods.

### 2.5 Related Work

The goal of this article is to study the application of QSS methods to Smart Grid problems described in the Modelica language. To the best of the authors' knowledge, this problem has not previously been studied.

The application of QSS methods to Modelica models was studied in [3, 10, 11], showing the benefits of QSS methods for problems with frequent discontinuities. Also the use of QSS methods (not using Modelica) for a large hybrid sparse load management problem was studied in [18].

The use of the Modelica language for Energy Management problems was studied in [6, 7, 8, 19, 20, 21] showing the powerful advantages that the language offers to this set of models.

An advanced control system for the optimal energy management of a building cooling system is studied in [5]. In this system, a centralized facility produces chilled water that is then distributed among a certain number of thermal zones (e.g. small houses or apartments). The optimal control algorithm requires multiple simulations of the whole system model for different parameter settings. A simplified, equation-based version of that model is presented in this paper. Although the original system was designed for a reasonably low number of users (5 to 20, i.e., a micro-grid), this simplified model is representative of a larger class of systems with a centralized heat or cooling source, and many end users with their independent control systems. Such systems can easily scale up to contain hundreds, thousands, or even more individual units. The controllers have also been simplified in this work, as the achieved speed-up factor is not depending on the specific control laws, but rather on the efficient way the QSS algorithm exploits

the sparsity and weak coupling of the system model, combined with the efficient event handling.

### 3 Case Studies

In this section we present two case studies. First a District Cooling System and second an Energy Market model. As stated before, the Modelica language is suitable for describing multi-physics and multi-energy problems like SG models. In fact there are Modelica libraries for modeling energy problems that help the development process.

The QSS stand-alone solver accepts models described in a subset of the Modelica language, therefore all models used in the article are coded in this simplified language called  $\mu$ -Modelica. Being a subset of the complete language,  $\mu$ -Modelica is accepted by all Modelica simulation tools enabling us to simulate the same model both in OpenModelica and in the QSS stand-alone solver. For more information regarding the transformation of Modelica to  $\mu$ -Modelica models, we refer the reader to [3].

#### 3.1 Case Study I: A District Cooling System

As mentioned above, the District Cooling System is adapted from [5] and consists of the following elements:

- The *Cooling Plant* that generates cooling power used to control the temperature of a cooling load.
- The *Chilled Water Circuit* that connects the cooling plant to the cooling load allowing heat transfer between the two.
- The *Cooling Load* that transfers heat to the chilled water circuit. The load is composed by a group of zones affected by heat exchange with the outside ambient and by internal heat gains such as occupants and office equipment. The chilled water circuit exchanges heat with the zones by means of fan coils.
- The *Chilled Water Temperature Controller* that keeps circuit temperature at a specified set-point. The control variable is the cooling plant cooling power set-point.
- The *Zone Temperature Controller* that keeps each zone at the desired temperature. The control variable is the fan coil valve opening.

In the following paragraphs, the model adopted for each element is described.

**Cooling Plant** The cooling plant is simplified in this article to deliver exactly the energy needed for the Chilled Water Circuit set-point  $Q_{CSP}$ .

**Chilled Water circuit** Chilled water circuit dynamics are described using a lumped RC model. The following power balance equation can be written:

$$C_{CW} \frac{dT_{CW}}{dt} = \sum_i^N (Q_{ZAi}(t) - Q_C(t))$$

where  $T_{CW}$  is the circuit temperature and  $C_{CW}$  its thermal capacity.  $Q_{ZAi}$  is the heat exchanged with the  $i$ -th zone and  $Q_C(t)$  is the cooling power contribution provided by the Cooling Plant. Heat losses in the circuit are neglected.

**Cooling Load** Zones are modeled as lumped RCs as well. Their power balance equation is:

$$C_{ZA} \frac{dT_{ZAi}}{dt} = -Q_{ZAi} + k_{out}(T_{OA}(t) - T_{ZAi}(t)) + Q_{INTi}(t)$$

where  $T_{ZAi}$  is the zone temperature and  $C_{ZA}$  its thermal capacity,  $Q_{ZAi}$  is the heat exchanged with the chilled water circuit,  $X_{C,Zi}$  is the heat exchanger valve opening, and  $Q_{INTi}$  is the heat produced by zone occupants.  $Q_{ZAi}$  evolves according to the following expression:

$$\tau_{ex} \dot{Q}_{ZAi}(t) + Q_{ZAi}(t) = X_{C,Zi}(t) k_{cw}(T_{ZAi}(t) - T_{CW}(t))$$

where  $\tau_{ex}$  is the heat exchanger time constant. It is worth noticing that the introduction of the exchanger dynamics has the twofold purpose of a more accurate modeling and of obtaining a stiff model in order to test the QSS solver's performance in such conditions.  $Q_{INTi}$  is modeled according to the following polynomial function of the zone temperature:

$$Q_{INTi}(t) = (p_1 T_{ZAi}^2(t) + p_2 T_{ZAi}(t) + p_3) n_{peoplei}(t),$$

where  $n_{peoplei}$  is the number of zone occupants. Such a model is proposed in [2] where suitable coefficient values  $p_1$ ,  $p_2$  and  $p_3$  can be found.

The number of occupants is generated by a simple stochastic model: the next arrival/departure time of a person is given by a fixed time (1000 seconds) plus a uniform random variate between 0 and 1000 seconds. At each event, a person either comes in or leaves with

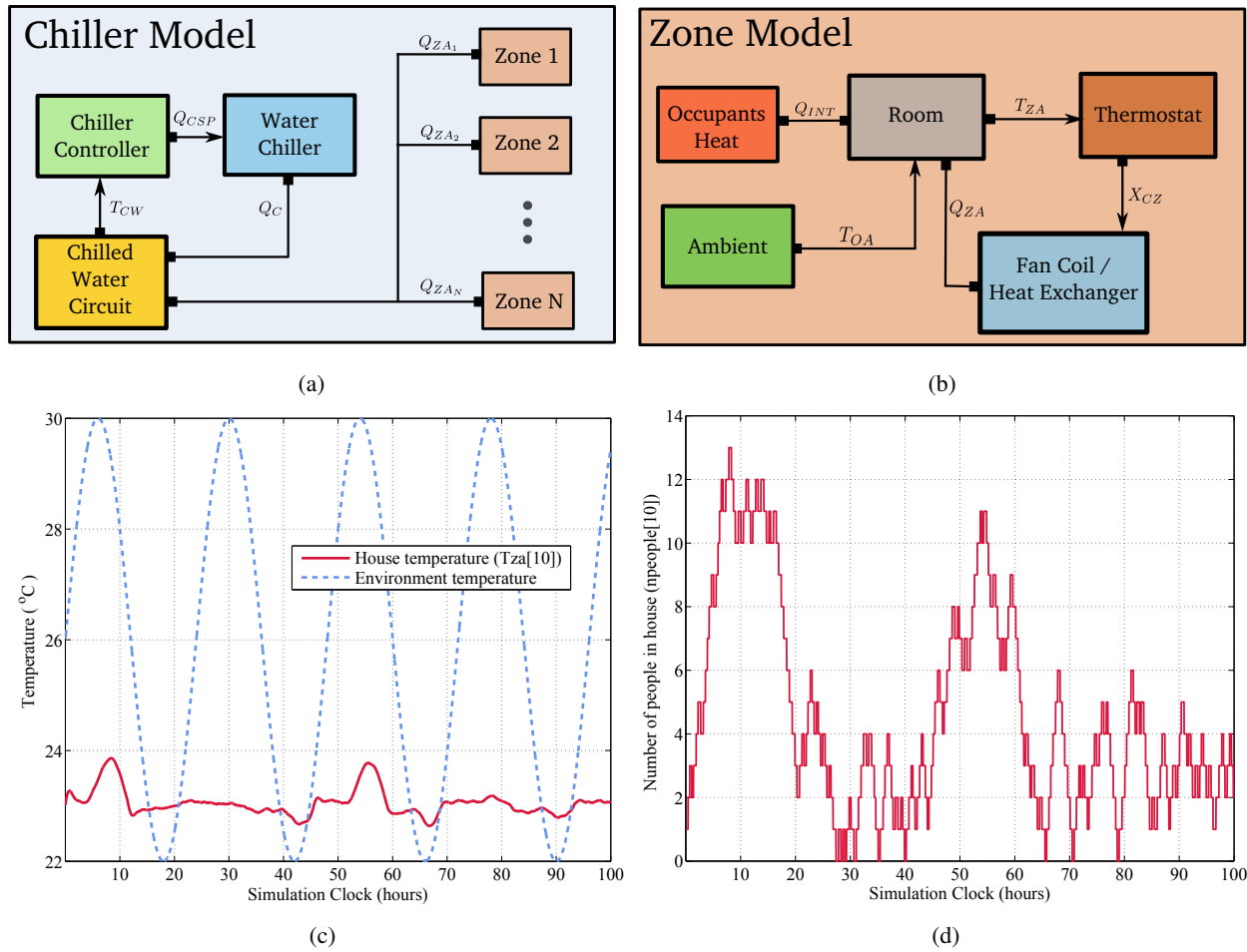


Figure 1: District Cooling System model graphical representation (a) and, in more detail, the submodel used for each zone (b). In (c) the simulated trajectory of the temperature state variable in zone 10 is plotted against the ambient temperature, while in (d) the number of the people being present in zone 10 is depicted over time.

a 50% probability. Although this model does not represent any specific realistic pattern for the coming and going of occupants, it serves the purpose of generating a number of *uncorrelated* time events, whose density in time grows with the number of zones. Any realistic system will exhibit this kind of behavior, as events happening in different buildings will usually be uncorrelated, no matter what their actual probability distribution is.

**Chilled Water Temperature Controller** A PI controller is adopted to control chilled water circuit temperature. The control variable is the power set-point for the cooling plant  $Q_{CSP}$ . Since no dynamics are considered for the chillers, the power set-point coincides with the actual power generated. The controller

is modeled as follows:

$$\begin{aligned} \dot{z}_{C,CW}(t) &= -\frac{k_{I,CW}}{k_{P,CW}} \cdot z_{C,CW}(t) + \frac{k_{I,CW}}{k_{P,CW}} \cdot Q_{CSP}(t) \\ Q_{CSP}(t) &= \Phi_{[0, Q_{C,max}]}(k_{P,CW} \cdot e_{C,CW}(t) + z_{C,CW}(t)) \\ e_{C,CW}(t) &= T_{CW}(t) - T_{CWSP}(t) \end{aligned}$$

where  $z_{C,CW}(t)$  is the integral state,  $k_{P,CW}$  and  $k_{I,CW}$  are the PI gains, and  $\Phi_{[a,b]}(\cdot)$  is the saturation function:

$$\Phi_{[a,b]}(x) = \begin{cases} a, & x < a \\ x, & x \in [a, b] \\ b, & x > b. \end{cases}$$

**Zone Temperature Controller** Each zone temperature is controlled by a PI controller as well. The control variable is the heat exchanger valve opening

$X_{C,Z}$ , spanning the range  $[0, 1]$ .

$$\begin{aligned}\dot{z}_{C,Z}(t) &= -\frac{k_{I,Z}}{k_{P,Z}} \cdot z_{C,Z}(t) + \frac{k_{I,Z}}{k_{P,Z}} \cdot X_{C,Z}(t) \\ X_{C,Z}(t) &= \Phi_{[0,1]}(k_{P,Z} \cdot e_{C,Z}(t) + z_{C,Z}(t)) \\ e_{C,Z}(t) &= T_{ZA}(t) - T_{ZASP}(t),\end{aligned}$$

**Model scaling** The presented model is designed to scale with the number of zones  $N$ , while providing a reasonable behavior for all controlled variables. For this purpose, certain model parameters are proportional to the number of zones  $N$ . In particular, the cooling plant maximum power  $Q_{C,max}$  and the cooling plant controller (chilled water temperature controller) gains  $k_{I,CW}$  and  $k_{P,CW}$  are proportional to  $N$ . The chilled water circuit thermal capacity  $C_{CW}$  is also linearly scaled with the number of zones  $N$ .

### 3.2 Case Study II: An Energy Market

The Energy Market model was introduced in [6] as a typical toy model written in Modelica and capturing many of the aspects typically found in realistic smart grid applications. The interactions between the different components in the Energy Market model are graphically sketched in Fig. 2. The model consists of the following components:

**Environment** We assume that the temperature of the environment is given by a sinusoidal function:

$$T_{amb} = \overline{T_{amb}} + \Delta T \sin(\omega t + \phi)$$

where the mean temperature  $T_{amb}$  is set to  $10^\circ\text{C}$ , while the frequency and offset are selected such that the minimum temperature is reached every midnight.

**Heaters** Each house has a heater that is controlled by an agent that switches it on and off, according to:

$$\dot{Q}_i^{heater} = \begin{cases} 0 & \text{if } T_i > T_i^{max} \\ P_{heat} & \text{if } T_i < T_{min} \end{cases} \quad (4)$$

**Walls** Each house has one wall that acts as a thermal resistor with the heat flow given by:

$$\dot{Q}_i^{wall} = \frac{1}{R_{th}}(T_i - T_{amb}), \quad (5)$$

where  $R_{th}$  is the thermal resistance of the wall.

**Windows** We assume that each house has one window that exhibits a stochastic behavior. More specifically, we assume that the opening time of each window is drawn randomly from a uniform distribution. It is closed again a random amount of time later.

$$\begin{aligned}openNextT[i] &\sim \mathcal{U}(pre(openNextT[i]) + 1000, 50) \\ closeNextT[i] &\sim \mathcal{U}(openNextT[i] + 100, 200),\end{aligned}$$

Each time a window is open, heat is exchanged between the environment and the house according to:

$$\dot{Q}_i^{window} = G(T_i - T_{amb}), \quad (6)$$

where  $G$  is a large heat conductance constant.

**Agents** Each house has a simple controller that controls the heater settings optimizing the power consumption. The agent turns the heater on at a lower goal temperature  $T_{min}$  and turns it off at an upper temperature  $T_{max}$ . If the energy price calculated in the energy market exceeds a threshold  $p_{max}$ , the agent decreased the upper level  $T_{max}$  to  $T_{max}^{alt}$ .

$$T_i^{max} = \begin{cases} T_{max} & \text{if } p < p_{max} \\ T_{max}^{alt} & \text{if } p \geq p_{max} \end{cases} \quad (7)$$

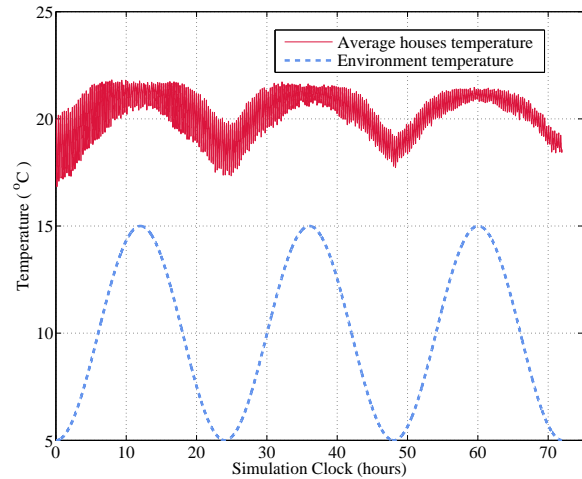


Figure 3: Simulated trajectories of the average temperature in the houses against the ambient temperature.

**Houses** Houses act as energy consumption units. The temperature inside each house is related to the heat flows described in the previous paragraphs with the following formula:

$$\dot{T}_i = \frac{1}{\rho V C_{th}}(\dot{Q}_i^{heater} - \dot{Q}_i^{window} - \dot{Q}_i^{wall}) \quad (8)$$



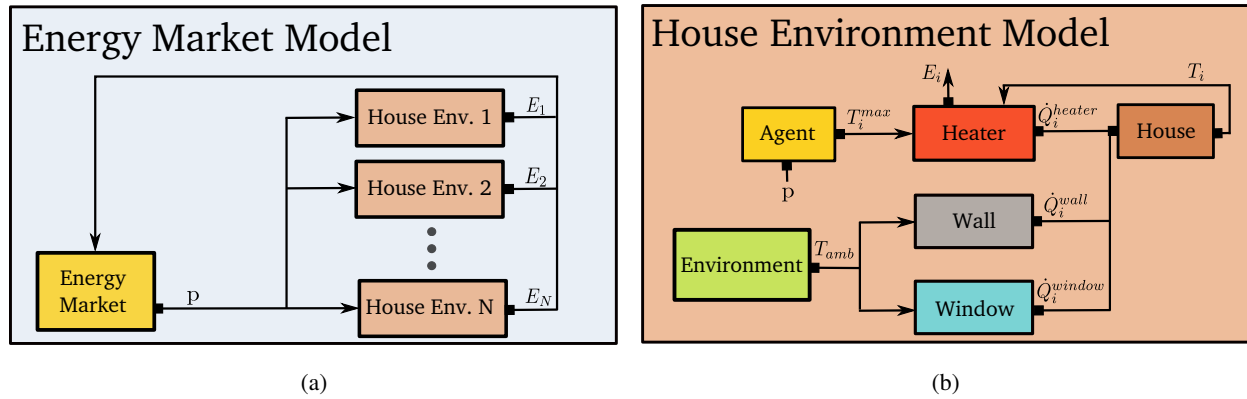


Figure 2: Energy Market model graphical representation (a) and, in more detail, the submodel used for each house(b)

The energy consumed per unit of time for a specific house is the power of its heater integrated over a time unit, as given by:

$$E_i = \dot{Q}_i^{heater} \cdot t_{unit} \quad (9)$$

**Energy Market** The energy market component simulates the behavior of an energy price regulator for the whole network. According to the estimated energy price the house agents decide if they should reduce the energy consumption or not. Various energy price models could be employed, but for simplicity we choose to linearly relate the energy price to the mean energy consumed in the houses:

$$p = \bar{p} + p_1 \times \frac{1}{N} \sum_{i=1}^N E_i$$

## 4 Results

In this Section we show the simulation results for the two models presented before. Both  $\mu$ -Modelica models can be downloaded from [1]. We compare the runtime efficiency and quality of the solutions obtained by different integration methods for different system sizes and tolerance values.

### Simulation Benchmark

The simulation benchmark is as follows:

- Runge-Kutta and DASSL results were computed using OpenModelica 1.9.1 (r18381) (RML version).
- LIQSS2,3 and QSS3 results were computed using the QSS Stand Alone Solver from [9] r645.

- The simulation platform is a Dell 32bit desktop with a quad core processor @ 2.66 GHz and 4 GB of RAM.
- The Jacobian matrices of the presented models are quite sparse and banded. This information could be exploited by all algorithms to make the simulation more efficient, however this has not been investigated for DASSL and Runge-Kutta methods. The QSS methods exploit this fact natively without having to get any information on the structure of the Jacobian matrix.

Calculating the accuracy of the simulations can only be performed approximately, since the state trajectories of the models cannot be computed analytically. To estimate the accuracy of the simulation algorithms for a given setting, reference trajectories ( $\mathbf{t}^{\text{ref}}, \mathbf{y}^{\text{ref}}$ ) were obtained using LIQSS3 with a tight tolerance of  $1 \cdot 10^{-9}$  on an equidistant grid consisting of 5000 points. To calculate the simulation error, all methods were forced to output points on the same equidistant grid, without changing the integration step, thus obtaining simulated trajectories ( $\mathbf{t}^{\text{ref}}, \mathbf{y}^{\text{sim}}$ ). Then, the mean absolute error is calculated as:

$$error = \text{mean}(|\mathbf{y}^{\text{sim}} - \mathbf{y}^{\text{ref}}|). \quad (10)$$

Regarding the error calculation we have to note that special care had to be taken in order to achieve comparable solutions from two independent runs of each model, since both models are based on the generation of random event sequences. To this end, we implemented a special random generator that, at every call, outputs the seed for its next call. Therefore, starting from the same seed, two independent model simulations generate the exact same sequence of random events.

The measured CPU time (simulation time) should not be considered as an absolute ground-truth since it will vary from one computer system to another, but the scaling of the algorithms as well as their relative ordering is expected to remain the same. Another important aspect is that, in order to objectively compare the simulation time needed by different algorithms we did not compare the time measurements of the algorithms for the same requested tolerance, but for the same achieved error.

#### 4.1 Case Study I

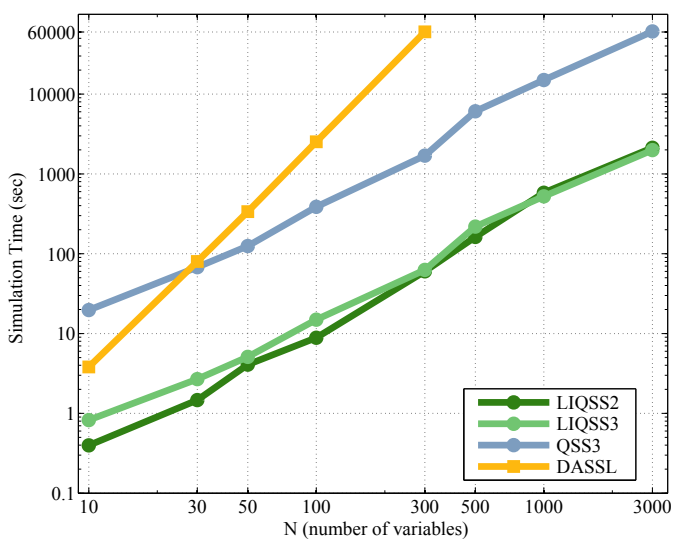


Figure 4: Simulation time for varying size of the District Cooling System model. All algorithms achieve a mean error on the order of  $10^{-4}$ .

The District Cooling System model is a comparatively stiff model and this becomes apparent when comparing the measured CPU time of the QSS3 and LIQSS methods in Fig. 4. This is confirmed by estimating the different time constants. The time constants of the temperature controlled zone ( $\sim 3,8 \times 10^3$  sec) and of the temperature controlled chilled water circuit ( $\sim 7,5 \times 10^3$  sec) proved to be greater by three orders of magnitude than the time constant of the heat exchanger ( $\sim 1$  sec).

Regarding the scaling of the algorithms, Fig. 4 suggests that DASSL scales quadratically ( $\sim 6N^2$ ), while QSS methods scale linearly with the number of variables ( $\sim 3N$ ). Due to the time constraints for the present study we had to stop the DASSL experiments at  $N=300$ , but the linear scaling of the QSS methods allowed us to test their performance up to  $N=3000$ , thus for a 10 times larger model.

Besides time, another factor that prohibited us from performing further experiments with DASSL on larger models, was that the OpenModelica compiler failed to compile larger models. For the largest model, that all methods could simulate ( $N=300$ ), the LIQSS methods are more than two orders faster than DASSL ( $10^2$  sec compared to  $600 \times 10^2$  sec).

Finally, a very interesting and useful aspect of the QSS methods is that they exhibit a strong correlation between the requested tolerance and the achieved error. This correspondence is depicted in Fig. 5a for the LIQSS2 method (the other QSS methods exhibit similar behavior). In contrast, the performance of DASSL was only slightly affected by changing the requested tolerance. This is a very important feature of the QSS methods as it allows the user to exploit the trade-off between computational speed and achieved error. Indeed in Fig. 5b, we see that a user who is willing to sacrifice one order of simulation accuracy will be rewarded by a simulation that executes roughly ten times faster when using LIQSS2.

#### 4.2 Case Study II

As the Energy Market model is not stiff we simulated it in OpenModelica using both DASSL and the fourth-order explicit Runge-Kutta algorithm. The measured simulation timings are shown in Fig. 6 where all methods achieve a mean error of order  $10^{-5}$ . The scaling of the algorithms, as well as their relative performance, agrees with the one obtained for the District Cooling System model. However, all methods perform better on this benchmark, because it is a simpler model in general.

More precisely, DASSL scales quadratically with the number  $N$  of variables ( $\sim 2N^2$ ) while Runge-Kutta scales linearly ( $\sim 5N$ ) since it is an explicit algorithm and does not have to perform any matrix inversion calculations. All three QSS methods exhibit a linear scaling ( $\sim N$ ) with the explicit QSS3 being marginally faster than the LIQSS3 algorithm (QSS3 needed 250 sec for  $N=10000$ , while LIQSS3 270 sec). For  $N=300$ , the LIQSS methods are over three orders more efficient than DASSL and over two orders more efficient than Runge-Kutta (10 sec compared to  $5000 \times 10$  sec and  $180 \times 10$  sec respectively).

Besides the linear scaling of the QSS methods becomes prominent their advantage over classical methods when simulating large sparse hybrid models with discontinuities. Each variable is being updated locally at its own speed, with no need of making global computations on the whole system matrix. Further-

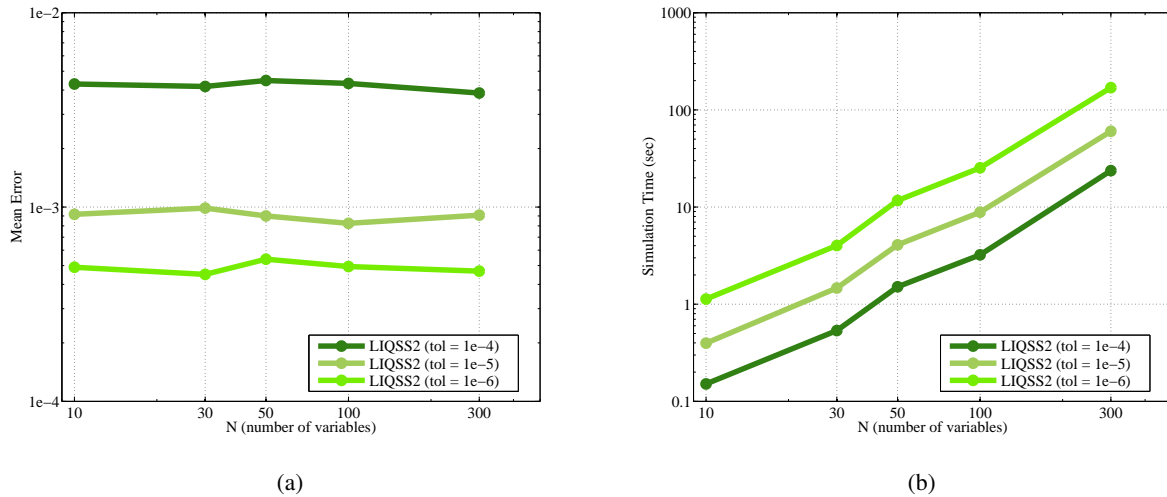


Figure 5: District Cooling System model - Mean Simulation Error (a) and Simulation Efficiency (b) for LIQSS2 and varying requested tolerances.

more, discontinuities are being handled as native simulation steps without the need of backtracking to detect the zero-crossings. Therefore, we observe that even though Runge-Kutta methods scale linearly with the system order just like the QSS methods, the latter are much more efficient than the former. QSS methods can simulate a system with  $N=10000$  states within the same execution time as a Runge-Kutta algorithm needs to simulate a much smaller system with  $N=300$  states.

## 5 Conclusion and Future Work

In this article we study the use of Quantized State System (QSS) integration methods for Smart-Grid (SG) simulation problems. The QSS methods have certain features (intrinsic sparsity exploitation, semi-implicit stiffness treatment and efficient discontinuity handling) that make them suitable for simulating SG models.

After analyzing two large hybrid SG models and comparing the efficiency and the quality of the solution obtained by the QSS methods against standard numerical integration methods we can conclude that:

- In both cases QSS methods outperform DASSL (and Runge-Kutta) by more than two orders of magnitude in terms of simulation speed, while at the same time, achieving a comparable simulation error.
- The QSS methods scale linearly with system size, while DASSL scales quadratically. The Runge-Kutta solvers also scale linearly, but they are far less efficient than their QSS competitors nevertheless.
- In both examples the QSS stand-alone simulator is able to handle larger model without running out of memory.
- We were able to simulate with the QSS methods up to 1000 times larger models than with DASSL, while still needing much less time to perform the simulations.

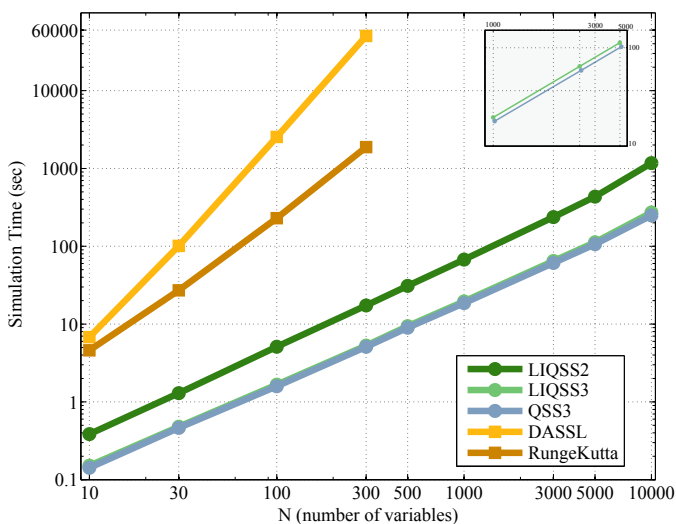


Figure 6: Simulation time for varying size of the Energy Market model. All algorithms achieve a mean error on the order of  $10^{-5}$ .

However, there still remain open problems to be addressed in the future. First of all, we need to perform experiments with a larger set of models typically used in the SG community, while at the same time testing more numerical integration methods against the QSS family. We note here that other implicit methods included in the OpenModelica environment, such as Radau and Lobatto, have been tested but failed to simulate the models. A necessary step that has to be performed in the future is including the family of QSS methods as integration methods in OpenModelica.

Finally, an interesting line of research could be the utilization of QSS methods in energy optimization algorithms, such as the one proposed in [5].

## 6 Acknowledgments

This work was in part funded by CTI grant Nr.12101.1;3 PFES-ES and supported by the OPENPROD-ITEA2 project. The authors would like to thank the developer of the stand-alone QSS solver, Joaquín Fernández, for fixing several bugs that enabled us to correctly and efficiently simulate the analyzed models.

## References

- [1] <http://people.inf.ethz.ch/florosx/modelica2014/>.
- [2] *CIBSE Guide A: Environmental Design*. CIBSE Publications, Norwich, UK, 2006.
- [3] F. Bergero, X. Floros, J. Fernández, E. Kofman, and F. E. Cellier. Simulating Modelica models with a Stand-Alone Quantized State Systems Solver. In *9th International Modelica Conference*, 2012.
- [4] F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, New York, 2006.
- [5] N. M. Ceriani, R. Vignali, L. Piroddi, and M. Prandini. An approximate dynamic programming approach to the energy management of a building cooling system. In *European Control Conference, Zurich (Switzerland), July 17-19*, pages 2026–2031, 2013.
- [6] A. Elsheikh, E. Widl, and P. Palensky. Simulating complex energy systems with modelica: A primary evaluation. In *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*, pages 1–6, 2012.
- [7] F. Felgner, S. Agustina, R. C. Bohigas, R. Merz, and L. Litz. Simulation of thermal building behaviour in modelica. In *2nd International Modelica Conference*, March 2002.
- [8] F. Felgner, R. Merz, and L. Litz. Modular modelling of thermal building behaviour using modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12(1):35–49, 2006.
- [9] J. Fernández and E. Kofman. Implementación autónoma de metodos de integración numérica QSS. Technical report, FCEIA - UNR, Rosario, Argentina, 2012.
- [10] X. Floros, F. Bergero, F. E. Cellier, and E. Kofman. Automated simulation of modelica models with qss methods - the discontinuous case -. In *8th International Modelica Conference*, March 2011.
- [11] X. Floros, F. Cellier, and E. Kofman. Discretizing time or states? a comparative study between dassl and qss. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT, Oslo, Norway, October 3, 2010*, pages 107–115, 2010.
- [12] P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling, Simulation, and Development Environment. *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.
- [13] P. Fritzson and V. Engelson. Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *ECOOP*, pages 67–90, 1998.
- [14] E. Kofman. Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25(5):1771–1797, 2004.
- [15] E. Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. *Latin American Applied Research*, 36(2):101–108, 2006.
- [16] E. Kofman and S. Junco. Quantized State Systems. A DEVS Approach for Continuous

- System Simulation. *Transactions of SCS*, 18(3):123–132, 2001.
- [17] G. Migoni, M. Bortolotto, E. Kofman, and F. E. Cellier. Linearly implicit quantization-based integration methods for stiff ordinary differential equations. *Simulation Modelling Practice and Theory*, 35:118 – 136, 2013.
- [18] C. Perfumo, E. Kofman, J. Braslavsky, and J. Ward. Load Management: Model-Based Control of Aggregate Power for Populations of Thermostatically Controlled Loads. *Energy Conversion and Management*, 55:36–48, 2012.
- [19] A. Sodja and B. Zupancic. Modelling thermal processes in buildings using an object-oriented approach and modelica. *Simulation Modelling Practice and Theory*, 17(6):1143 – 1159, 2009.
- [20] M. Wetter. Modelica-based modelling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation*, 2(2):143–161, 2009.
- [21] M. Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *Journal of Building Performance Simulation*, 4(3):185–203, 2011.
- [22] B. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation - Second Edition*. Academic Press, 2000.



# On the Simulation of Offshore Oil Facilities at the System Level

Joris Costes<sup>1,2</sup> Jean-Michel Ghidaglia<sup>2</sup> Philippe Muguerra<sup>3</sup> Keld Lund Nielsen<sup>4</sup>  
Xavier Riou<sup>3</sup> Jean-Philippe Saut<sup>1</sup> Nicolas Vayatis<sup>2</sup>

<sup>1</sup>Eurobios SCB, 86 avenue Lénine, 94250 Gentilly, France  
{joris.costes, jpsaut}@eurobios.com

<sup>2</sup>CMLA, ENS Cachan, UMR 8536 CNRS, 61 avenue du président Wilson, 94235 Cachan cedex, France  
{jmg, vayatis}@cmla.ens-cachan.fr

<sup>3</sup>eni saipem, 7 avenue de San Fernando, 78180 Montigny-le-Bretonneux, France  
{Philippe.MUGUERRA,Xavier.RIOU}@saipem.com

<sup>4</sup>eni spa, exploration & production division, San Donato Milanese (Mi), Italy  
Keld.Lund.Nielsen@eni.com

## Abstract

Offshore oil facilities are complex systems that involve elaborate physics combined with stochastic aspects related, for instance, to failure risk or price variation. Although there exist many dedicated software tools to simulate flows typically encountered in oil exploitations, there is still no tool that combines physical (mostly engineering fluid mechanics) and risk simulation. Such a tool could be useful to engineers or decision makers for specification, design and study of offshore oil facilities. We present a first step towards the creation of such a tool. Our current simulator is based on new Modelica components to simulate fluid flows and on stochastic simulation at a higher level, for modeling risk and costs. Modelica components implement physical models for single and two-phase flows in some typical devices of an offshore field. The risk simulation uses Markov chains and statistical indicators to assess performance and resilience of the system over several months or years of operation.

*Keywords: fluid flow; two-phase flow; risk estimation; Monte Carlo simulation*

## 1 Introduction

With the increasing rarity of readily accessible reservoirs, as oil has to be extracted from deeper undersea, capital investments and risks associated to offshore oil facilities become higher and higher. In this context, a careful evaluation of cost and risk represents a crucial step in the conception of a new offshore facility. This evaluation must be performed at the system level because it involves phenomena of very different na-

tures and scales, that can interact together. We identify three main types of phenomena to be taken into account in the simulation: *Physical*, *failure-related* (risk) and *economic* (cost). The *physical phenomena* obviously include fluid flow through common components (pipes, tanks, valves, etc.) but can also include various other aspects like e.g. strength of materials, heat transfer or chemistry of wax or hydrate formation. The *failure phenomena* are the discrete-time events due to specific (extreme) conditions (e.g. break due to component fatigue) or to accidental situations. Simulating failures is useful to evaluate the potential risks of a particular architecture/design. The *economic phenomena* encompass all that is related to cost or price but does not directly depends on physics (unlike e.g. the oil production). For instance the oil barrel price, the price of pipe wall material (typically steel) or the transportation cost. All these variables directly impact the profitability of an exploitation.

This paper presents our first effort in the conception of a system-level offshore facility simulation. Before starting the work, it was necessary to point out simulation requirements and expected difficulties. Requirements are more related to the user point-of-view while difficulties are more related to the physics of an offshore oil facility:

- Reasonable computation times are preferable since we want the simulator to be usable as a design tool,
- Complexity: An offshore exploitation can involve dozens or hundreds of elementary components,
- Modularity: A model is meant to be built by as-

sembling elementary components of typical parts of an offshore field,

- Accurate prediction of highly-turbulent fluid flow is not possible (typical Reynolds number in oil & gas industry flows is  $> 10000$ ),
- Many physical parameters are known only in certain ranges or even uncertain,
- Experimental validation is difficult or limited to existing literature.

Henceforth it was decided to build first physical models with moderate complexity in order to follow the large-scale behavior of the real components within reasonable computation time. Another choice was to decompose the simulation into two different layers: One for the physics and one for the cost/risk estimation. The physics layer models deterministic phenomena only, while the cost/risk layer is based on probabilities to model risk and uncertainty. The physical simulation takes advantage of the flexibility and modularity that are possible with Modelica features (object-oriented, acausality). Other authors have used a similar approach of employing a stochastic simulation layer on top of a Modelica-based simulation. For instance in the context of Building Performance Simulation [14] to model weather and room occupancy as stochastic processes. Propositions were recently submitted by Bouskela *et al.* [3] to enrich the Modelica language with the possibility to define uncertain variables with user-configurable probability law. The authors exposed some applications in power plant or combustion engine field to perform data reconciliation or uncertainty propagation. With the current Modelica version, the authors had to rely on the external program OpenTURNS [7] to compute uncertainty propagation on a fluid pipe system example. It seems presently unavoidable to work with this kind of architecture (Modelica + external program with an interface layer) since stochastic modeling is clearly out of the scope and objectives of the current Modelica specifications.

The paper is divided as follows. Section 2 focuses on fluid flow simulation. It describes the hypotheses and equations used to build the Modelica components of the offshore facility. Section 3 deals with the estimation of cost and risk. It details the different variables of interest and their stochastic model (Markov chain representation and Monte Carlo simulation). Section 4 presents some first results obtained on a simplified architecture used as a proof of concept.

## 2 Simulation of flows in an offshore oil facility

The first choice when conceiving the fluid flow simulation was to decide the accuracy level that would give the appropriate balance between fast computation time and physical coherence. To ensure the computational tractability of the models we have chosen 1D or 0D models depending on the component. This for permanent-regime study. The latter choice comes from the considered time scale that is rather large in order to estimate typical daily production (hours extrapolated to a full day). In the current development stage, we are not yet concerned with heat transfer so isothermal transformations are assumed. The fluid that flows from an oil reservoir is usually a mixture of oil and gas (in particular because gas might be injected inside the reservoir to increase the flow rate of production). The basic connector used in our simulation is then defined by three parameters ( $p, q, \varphi$ ):

```
connector TPPort "Two-phase port"
  SIunits.Pressure pressure "Pressure";

  //Volumetric flow:
  flow SIunits.VolumeFlowRate q;

  //Volume ratio of liquid in the mixture:
  stream SIunits.VolumeFraction phi;
end TPPort;
```

In addition to the permanent-regime assumption, we assume that fluids are incompressible with exceptions for a few components (e.g. the oil-gas separator in Section 2.4). The aforementioned assumptions are more restrictive than the ones of the *Fluid* library [4] in the Modelica Standard Library that is more generic, at the price of a higher computation time.

### 2.1 Single-phase flow in a pipe

The model of single-phase flow in a pipe is encountered only in limited areas of the offshore field, e.g. after an oil-gas separator. It is however useful in order to compute the virtual single-phase pressure drop used in the two-phase pipe model (Section 2.2). It requires a variant of the TPPort connector where  $\varphi$  (phi) is removed. The fluid velocity  $v$  is computed from the volume flow rate  $q$  and the pipe cross-sectional area  $A$ :

$$v = \frac{q}{A}. \quad (1)$$



Since we have volume flow conservation,  $q = q_a = q_b$  where  $q_a$  and  $q_b$  are the values of  $q$  in the two connectors at the ends of the pipe.

Reynolds number is

$$Re = \frac{\rho D_h |v|}{\mu}, \quad (2)$$

where  $\rho$  is the fluid density,  $D_h$  the pipe hydraulic diameter and  $\mu$  denotes dynamic viscosity of the fluid.

The frictional pressure loss is computed as

$$\Delta p = \pm f_d \frac{L}{D_h} \rho \frac{v^2}{2}, \quad (3)$$

where  $f_d$  is the Darcy friction factor and  $L$  the pipe length. The sign of  $\Delta p$  must be chosen so that the pressure decreases in the direction of the flow.

The Darcy friction factor  $f_d$  depends on the flow regime. As only turbulent flows are encountered in the considered oil and gas applications, only the turbulent regime is of interest. For this regime, among the many existing correlations, we chose Haaland's formula [9]:

$$f_{d_{turbulent}} = \left( -1.8 \log_{10} \left( \frac{6.9}{Re} + \left( \frac{k_s}{3.7 D_h} \right)^{1.11} \right) \right)^{-2}, \quad (4)$$

where  $k_s$  is the roughness height that characterizes the rugosity of the pipe inner wall. It is typically between  $1 \mu\text{m}$  and  $1 \text{mm}$ .

## 2.2 Two-phase flow in a pipe

Several Modelica models have been proposed to deal with two-phase flow modeling [1, 6, 10, 2], with applications to steam generators or refrigerators. These models are centered on accurate simulation (1D, boundary model) of a few components. To simulate an offshore field architecture, since we are for now only concerned with the related evolution of pressure loss and flow rate, we chose a much simpler model with very low computational requirements, based on the work by Lockhart and Martinelli [12]. Lockhart and Martinelli proposed a correlation to compute the pressure drop of a two-phase mixture in a pipe, from the pressure drops computed for the two (virtual) single-phase flows. Chisholm [5] gave some theoretical basis for the correlation and recommended a simplified version of the formula, for engineering calculations:

$$\Delta p_{TP} = \pm \left( \Delta p_L + C \sqrt{\|\Delta p_L \Delta p_G\|} + \Delta p_G \right), \quad (5)$$

where:

- $\Delta p_{TP}$  is the pressure drop for the two-phase mixture,
- $d\Delta p_L$  is the pressure drop as if the liquid flowed alone,
- $d\Delta p_G$  is the pressure drop as if the gas flowed alone,
- $C$  is a correction coefficient which depends on the flow type of each phase (see Table 1). In practice, only the turbulent-turbulent case is of interest.

The sign of  $\Delta p$  must be chosen so that the pressure decreases in the direction of the flow.

Flow regime		
Liquid	Gas	Coefficient $C$
turbulent	turbulent	20
laminar	turbulent	12
turbulent	laminar	10
laminar	laminar	5

Table 1: Coefficient  $C$  for two-phase pressure drop computation (from [5]).

Note that the library *FluidDissipation*[16] also refers to the work of Chisholm, in a more complete implementation<sup>1</sup>.

## 2.3 Junctions

They are components used to direct the flows in two pipes into a single one. The junctions can be for instance used to inject gas in a liquid flow in order to increase its flow rate. When the two mixing fluids are in the same phase, a simple model can be used that just averages their characteristics. When the two phases are different a finer model is necessary. We consider here the cases of horizontal and vertical junctions.

### 2.3.1 Horizontal junction

The configuration of Figure 1 is considered.  $(p_1, v_1, \rho_1)$ ,  $(p_2, v_2, \rho_2)$  and  $(p_3, v_3, \rho_3)$  are the pressure, velocity and density at the liquid inlet, gas inlet and mixture outlet respectively,

Fluid velocities are  $v_1 = \frac{q_1}{A}$ ,  $v_2 = \frac{q_2}{A_g}$  and  $v_3 = \frac{q_3}{A}$  where  $A$  is the cross-section area of the liquid inlet and mixture outlet and  $A_g$  is the cross-section area of the gas

<sup>1</sup><http://xrg-simulation.de/en/products/xrg-library/xrg-fluiddissipation-library>

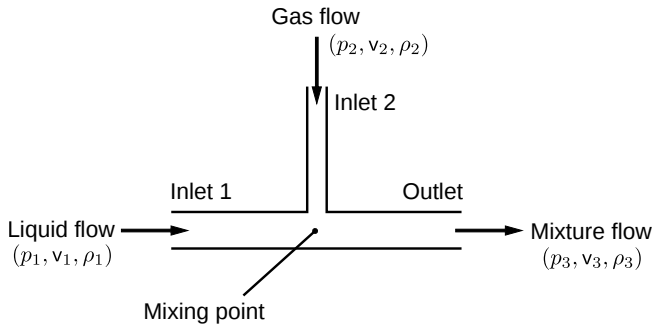


Figure 1: Schematic of the horizontal two-phase junction.

inlet. Conservation of both mass and volume flows gives:

$$\varphi = \frac{v_1 A}{v_1 A + v_2 A_g}, \quad (6)$$

$$\rho_3 = \varphi \rho_1 + (1 - \varphi) \rho_2, \quad (7)$$

$$v_3 = -(v_1 A + v_2 A_g) / A. \quad (8)$$

From Ji *et al.* [11] we took the empirical equation for the momentum correction coefficient  $K$  that is computed from the momentum flux ratio  $M$ . The equations are:

$$M = \frac{\rho_1 v_1^2}{\rho_2 v_2^2}, \quad (9)$$

$$K = 1 + 0.256 M^{0.223}, \quad (10)$$

$$p_1 - p_3 = K (\rho_3 v_3^2 - \rho_1 v_1^2). \quad (11)$$

Note that we use here the most generic formula in [11], because the roles of gas (side inlet) and liquid (front inlet) are swapped compared to what is in the paper. At this point, the system is under-determined since  $p_2$  does not appear in any of the above equations (it would if the junction angle was  $\neq 90^\circ$ ) so we need an extra equation. We choose the assumption that both inlet 1 and inlet 2 are close enough to the mixing point in order to have

$$p_1 = p_2. \quad (12)$$

### 2.3.2 Vertical junction

A vertical junction is a horizontal junction rotated with a  $90^\circ$  angle (Figure 2). Compared to the horizontal junction, a correction term is added to take into account the weight of the fluid:

$$p_1 - p_3 = K (\rho_3 v_3^2 - \rho_1 v_1^2 + 0.5 L g (\rho_1 + \rho_3)), \quad (13)$$

where  $g$  is the gravitational acceleration and  $L$  the junction length.

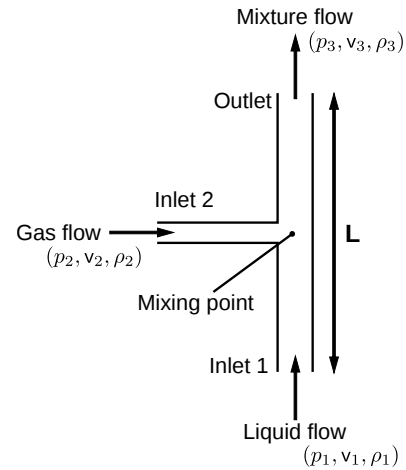


Figure 2: Schematic of the vertical two-phase junction.

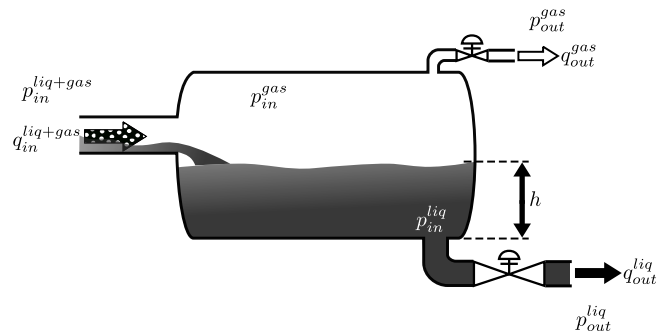


Figure 3: Schematic of the oil-gas separator.

## 2.4 Separator

The purpose of an oil-gas separator is to output two single-phase flows (one of liquid and one of gas) from one two-phase inflow (liquid+gas) (Figure 3). The studied separator dissociates the two phases by gravity, inside a tank or vessel. The physical input to the separator is the volume flow of oil-gas mixture that goes into the tank, while the outputs are the volume flows out of the tank. The device contains control loops to maintain the liquid level and the inside gas pressure at desired reference values. The oil-gas separator is consequently modeled as a controlled system. The two controlled values are the height of oil in the tank (denoted by  $h$ ) and the gas pressure inside the tank (denoted by  $p_{in}^{gas}$ ). Internal sensors (supposedly perfect) measure both  $h$  and  $p_{in}^{gas}$  so that they can be compared to their respective reference values  $h_{set}$  and  $p_{set}$ . There is one control loop for each of  $h$  and  $p_{in}^{gas}$  and some variables are involved in the two control loops [8] (Figure 4).

The controllers are simple PIDs (Proportional-Integral-Derivative). Transmission lines conduct the

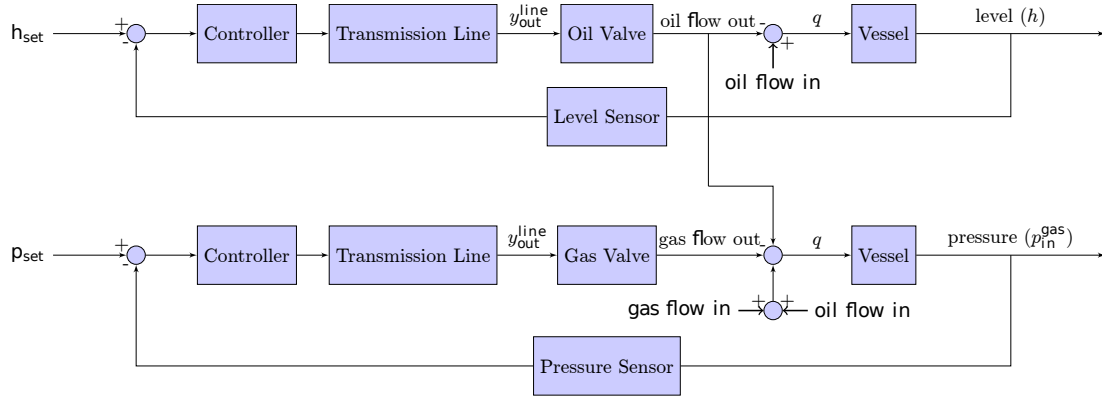


Figure 4: Liquid level and gas pressure control loops.

controller outputs to the control valves. They are modeled as first-order low-pass filters [8]. The valve aperture control parameter  $\sigma_{valve}$  is the line signal  $y_{out}^{line}$  after a  $[0; 1]$  saturation is applied:

$$\sigma_{valve} \in [0; 1] = \max(0, \min(y_{out}^{line}, 1)). \quad (14)$$

We give more details on each control loop in next sections.

#### 2.4.1 Level loop

The vessel is a horizontal cylinder with diameter  $d_{\varnothing}$ , radius  $r$  and length  $L$ . Let us introduce the distance  $\tilde{h}$  from the liquid height  $h$  to the vessel middle (i.e.,  $h=r$ ):

$$\tilde{h} = \begin{cases} h & \text{if } h \leq r, \\ d_{\varnothing} - h & \text{if } h > r. \end{cases}$$

The liquid level  $h$  is then related to liquid volume flows according to

$$q_{in}^{liq} - q_{out}^{liq} = 2\sqrt{d_{\varnothing}\tilde{h} - \tilde{h}^2} L \frac{dh}{dt}, \quad (15)$$

where

$$q_{in}^{liq} = \varphi q_{in}^{liq+gas}. \quad (16)$$

Finally  $q_{out}^{liq}$  is related to  $p_{in}^{liq} - p_{out}^{liq}$  with a friction law like in Section 2.1, which is denoted  $f_{fric}(\cdot)$ . The factor  $\sigma_{valve}$  is added because we assume the valve aperture to act linearly on the flow:

$$q_{out}^{liq} = \sigma_{valve} f_{fric}(p_{in}^{liq} - p_{out}^{liq}). \quad (17)$$

#### 2.4.2 Pressure loop

The gas volume flows are:

$$q_{in}^{gas} = (1 - \varphi) q_{in}^{liq+gas}, \quad (18)$$

$$q_{out}^{gas} = \sigma'_{valve} f'_{fric}(p_{in}^{gas} - p_{out}^{gas}), \quad (19)$$

where  $\sigma'_{valve}$  is the gas valve aperture and  $f'_{fric}(\cdot)$  a friction function like in Section 2.1. The ideal gas law in the vessel gives:

$$p_{in}^{gas} V^{gas} = n^{gas} RT, \quad (20)$$

where  $V^{gas}$  is the volume of gas inside the tank (above the liquid),  $n^{gas}$  is the gas mass quantity (moles) inside the vessel,  $R$  the ideal gas constant, and  $T$  the temperature inside the tank.

Differentiating Eq.(20), while replacing  $V^{gas}$  with  $(V^{total} - V^{liq})$ , leads to:

$$\underbrace{(V^{total} - V^{liq})}_{V^{gas}} \frac{dp_{in}^{gas}}{dt} - p_{in}^{gas} \frac{dV^{liq}}{dt} = RT(q_{in,mass}^{gas} - q_{out,mass}^{gas}), \quad (21)$$

where  $V^{total}$  is the tank volume,  $V^{liq}$  the volume occupied by the liquid,  $q_{in,mass}^{gas}$  and  $q_{out,mass}^{gas}$  are the gas mass inflow and outflow respectively. The tank volume is

$$V^{total} = \pi r^2 L. \quad (22)$$

Volume  $V^{liq}$  is computed from the liquid height by introducing

$$V_{tmp} = L \left( r^2 \arccos\left(1 - \frac{\tilde{h}}{r}\right) - (r - \tilde{h}) \sqrt{2r\tilde{h} - \tilde{h}^2} \right), \quad (23)$$

$$V^{liq} = \begin{cases} V_{tmp} & \text{if } h \leq r, \\ V^{total} - V_{tmp} & \text{otherwise.} \end{cases}$$

The mass flows are computed from the ideal gas law (also assuming  $p_{in}^{liq+gas} = p_{in}^{gas}$ ):

$$q_{in,mass}^{gas} = q_{in}^{gas} \frac{p_{in}^{liq+gas}}{RT}, \quad (24)$$

$$q_{out,mass}^{gas} = q_{out}^{gas} \frac{p_{in}^{gas}}{RT}. \quad (25)$$

Figure 5 shows the system response when the separator receives a sequence of inflow steps.

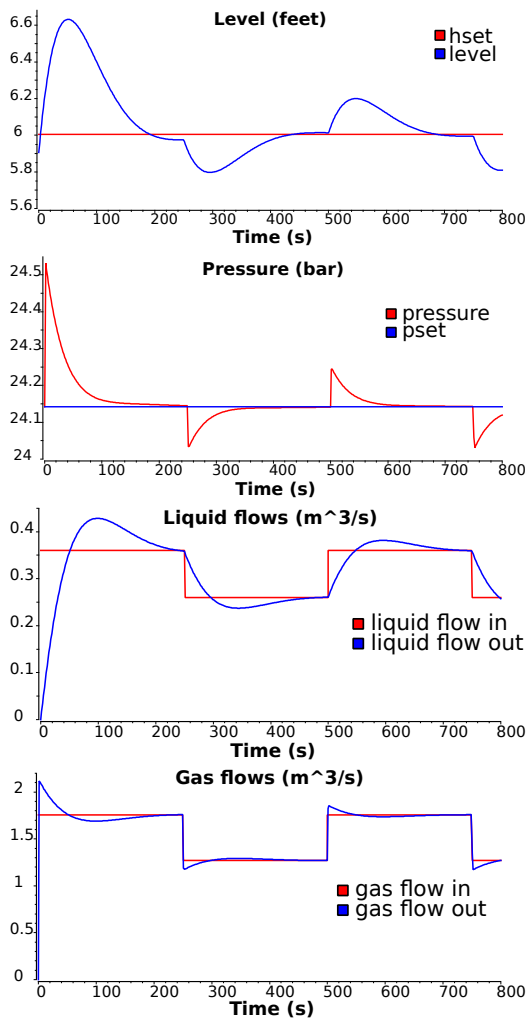


Figure 5: Responses to a square-wave input flow. The system shows a second-order behavior (overshooting and pseudo-oscillations).

## 2.5 Other models

Other models are necessary to simulate an offshore field. For instance, valves, pumps or tanks. The models are not described in detail here due to lack of space. They are chosen as simple as possible at this stage of the work. Hence ideal behavior is assumed:

- Valve: Two ports: port\_a, port\_b, one parameter: open=aperture signal:
 

```
equation
if noEvent(open >= 0 ) then
port_b.q = -port_a.q;
port_b.p = port_a.p;
port_b.phi = inStream(port_a.phi);
port_a.phi = inStream(port_b.phi);
```

```
else
port_a.q = 0;
port_b.q = 0;
end if;
```

- Pump: Two ports: port\_a, port\_b, one parameter: q0=imposed flow rate:
 

```
equation
port_a.q = q0;
port_b.q = -q0;
port_b.phi = inStream(port_a.phi);
port_a.phi = inStream(port_b.phi);
```
- Tank: One port: port\_in, one parameter: A=horizontal area, one variable: h=liquid level:
 

```
equation
//hydrostatic pressure:
port_in.p = rho * g * h;
der(h) = port_in.q/A;
```

## 3 Estimation of costs and risks

Estimating costs and risks requires a different modeling level than fluid flow simulation since it depends on exogenous factors, possibly stochastic (oil market, steel market, weather), or endogenous stochastic factors (failures). Dividing the simulation in two layers does not mean however that the layers are uncoupled. There is instead a strong dependency between them. For instance, the production income depends both on the extracted oil volume, computed in the physical simulation with an additional random perturbation term, and on the oil market and system state, the latter being modified by possible failures. Next sections describe the stochastic modeling of the facility. As further explained in Section 4, this part is not performed with Modelica but interacts with the Modelica simulation used for fluid flow modeling.

### 3.1 Stochastic model of the offshore oil facility

#### 3.1.1 Markov chain model

The system is assumed to be a Markovian process (i.e., memoryless) with discrete time. The time step is long (e.g. day or week) compared to the physical simulation time. The Markovian process is modeled as a Markov chain i.e., a finite state machine with transitions described as conditional probabilities. Each state of the chain corresponds to a particular operation state

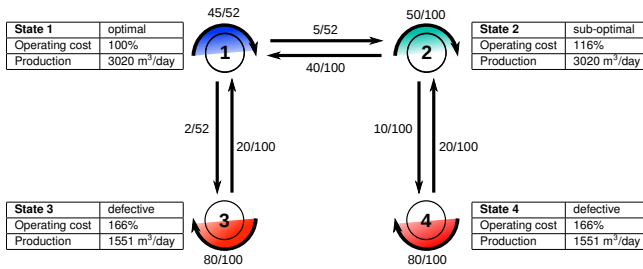


Figure 6: Markov chain representation of the offshore field with four possible operation states. The operating costs are given as percentages of the cost when in optimal state (state 1).

of the oil facility. Production volume of each state is computed with the physical simulation. It means that as many physical simulations will be conducted as the number of defined system states. The Markov chain requires the definitions of the possible transitions between states and their respective probability. These definitions are typically obtained from experimental data or expert knowledge. Figure 6 shows an example of Markov chain for four defined states with their associate operating cost and production volume. The Markov chain is used to simulate virtual life-cycles of the field under the randomness hypotheses on the events (price change, failure) that may occur over time. One such virtual cycle corresponds to one realization of randomness. Repeating many simulations allows to build empirical distributions of the output of interest (production, cost, resilience performance, etc.). This is the principle of Monte Carlo simulation.

### 3.1.2 Monte Carlo simulation

The goal of Monte Carlo simulation is to derive statistics from repeated simulations. With the Markov chain model, it means computing the evolution of the system state by random selection of the transition at each time step. The transition selection depends on the previously-defined transition probabilities. We now present some of the statistical quantities that are interesting in the context of offshore field simulation.

## 3.2 Assessment indicators

Various indicators can be calculated to obtain information about the system performance. Some indicators concern for instance how the production is affected by the failure events. We will present the resilience indicator that aims at quantifying the impact of failures on the production. Other indicators are related to the

gain and the risk of loss. Both types of indicator (production and gain) can be applied to one or many simulations. In the latter case, new indicators might be derived to estimate the uncertainty of the performance indicator, from its estimated variance for instance.

### 3.2.1 Production resilience

Resilience is the ability of a complex system to respond and recover from damages. The definition comes from ecology but can be found in various fields. For oil facility study, we will refer to the concept as defined in the study of urban resilience [13]. For a given simulation, resilience is used to quantify the effect of sub-optimal (i.e., disturbed) states on the level of performance. Resilience  $R$  is expressed in % and, for a discrete-time system, is computed as

$$R = 100 \left( 1 - \frac{(V_{opt} - V)}{V_{opt}} \right), \quad (26)$$

where  $V$  is the production volume for the considered time period and  $V_{opt}$  is a reference production volume that corresponds to optimal production state. Resilience will vary from 0% (no production) to 100% (optimal production). Repeated experiments give statistical values of duration and occurrences of disturbance situations.

### 3.2.2 Gain and risk of loss

The gain represents the difference between the value of recovered oil and the operation and capital costs:

$$\text{gain} = \text{recovered\_oil} \times \text{barrel\_price} \times \Phi_0 - \text{OPEX} \times \Phi_1 - \text{CAPEX} \times \Phi_2 - \text{risk\_losses}, \quad (27)$$

where OPEX stands for *operating expenditure* (i.e., ongoing cost to run the field), CAPEX stands for *capital expenditure* (i.e., cost to acquire or upgrade the equipments) and  $\Phi_0, \Phi_1, \Phi_2$  are factors due to income/cost sharing with consortium partners ( $\Phi_0, \Phi_1, \Phi_2$ ) or extra bank/insurance costs ( $\Phi_1, \Phi_2$ ).

The cumulative distribution function (cdf) of gain is denoted  $F_{\text{gain}}(x)$  and defined as

$$F_{\text{gain}}(x) = P(X_{\text{gain}} \leq x), \quad (28)$$

where  $P(X_{\text{gain}} \leq x)$  is the probability that the gain random variable  $X_{\text{gain}}$  is smaller than  $x$ . The considered gain is the gain over a given time period (e.g. a week or a year if several years are simulated). Risk indicators can be computed directly from the cdf. For instance, the probability of loss can be simply estimated

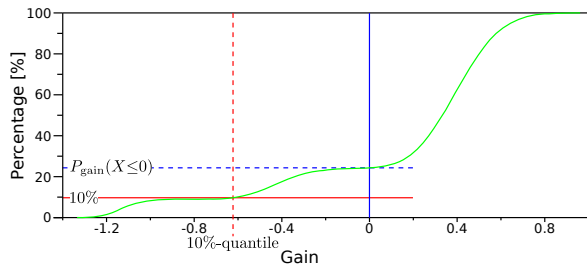


Figure 7: Estimation of the cumulative distribution function of the gain (green curve), from a one-year simulation. Intersection of the curve with the vertical line (Gain=0) gives an estimation of the probability of loss (blue dashed line). Intersection of the curve with the horizontal line at 10% gives an estimation of the first 10%-quantile (red dashed line). The gain is normalized so that the maximum weekly gain is 1.

as  $F_{gain}(0)$  and the  $k$ -th q-quantile ( $k \in [1..q]$ ) is the solution of  $F_{gain}(x) = \frac{k}{q}$ . Figure 7 shows the example of an estimated cdf with the estimated probability of loss and the estimated first 10%-quantile.

### 3.2.3 Uncertainty and performance

All the aforementioned indicators are estimations of the real quantities that could be obtained only from an infinite number of random paths. Therefore, all decisions made on behalf of the performance indicators must include some consideration about indicator uncertainties. A tradeoff between estimated performance and reliability of the estimation should be found; for instance by representing the indicator in the space (*performance, uncertainty*). Monte Carlo simulation already provides insights on the variability of indicators based on the gain cdf. Indeed, confidence intervals on risk indicators can be derived as shown in Figure 8. The uncertainty can be estimated in various manners, e.g. as a function of the variance of the performance estimator. The variance can be approximated with the results of several experiments. The number  $N$  of experiments can also quantify the reliability of an estimation since the latter increases with  $N$ .

## 4 Results on a simplified design

A simplified design has been chosen to illustrate the methodology (Figure 9). Oil is extracted from three wells whose flows are combined and sent to one among two vertical pipes (*risers*). The specific model for the riser is not detailed here. The two-phase flow is

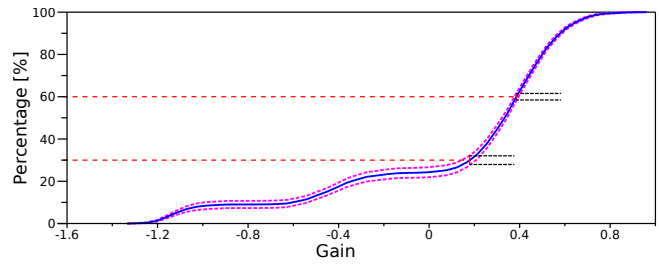


Figure 8: The cumulative distribution of gain averaged on 1000 one-year simulations (blue curve) and the confidence interval (magenta dashed curve) for a 30% and a 60% threshold. How far are the magenta curves from the blue curve tells how uncertain is the blue curve at a given time. The gain is normalized so that the maximum weekly gain –over the 1000 simulations– is 1.

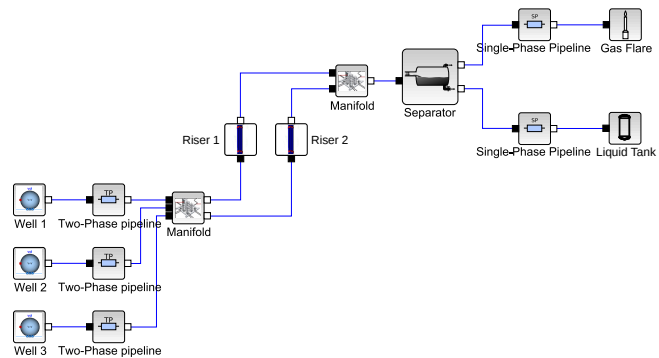


Figure 9: The studied simplified design of the offshore facility.

then received in an oil-gas separator and the two output single-phase flows are sent to a gas flare and an oil tank. The corresponding Markov chain is shown in Figure 6. States 1 and 2 are normal states with one of the two risers selected. States 3 and 4 are failure states (well 3 is blocked) with one or the other riser selected. Note that even though the characteristics (i.e., cost and production) of state 3 and 4 are identical, the states themselves are not as they do not have the same connections nor transition probabilities.

### 4.1 Fluid flow simulation

Some outputs of the fluid flow simulation are shown in Figure 10. In the presented results, initial oil level in the separator is far below the reference value, therefore all the input flow is used to fill the separator and the level in the tank does not increase until  $t \approx 2000$ s. Then, because of overshoot in the controlled system, level exceeds the reference value before eventually reaching the reference. A step is added to well flow

at  $t=5000$ s that explains the little peak in oil level.

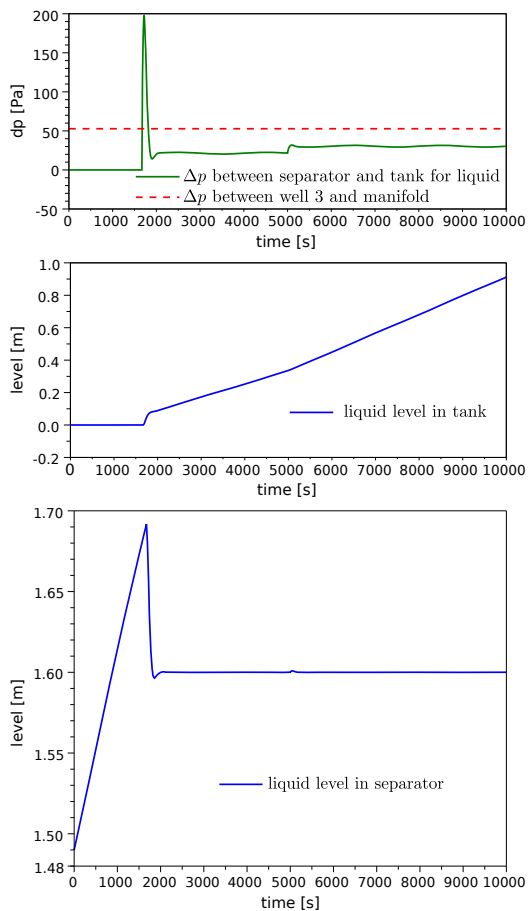


Figure 10: Some results of the fluid flow simulation with Modelica. From top to bottom: Pressure drops in a pipe with two-phase flow and at gas-oil separator outlet; oil level in final tank; oil level in separator.

## 4.2 Cost and risk simulation

Whereas all the fluid flow simulation is performed with Modelica, the cost and risk simulation uses Scilab [15]. Scilab has a built-in block diagram modeler/simulator called Scicos that can use blocks based on Modelica code. The block diagram of Figure 9 is from Scicos interface. The simulation can then be ran from a Scilab script using the `scicos_simulate(...)` command. Results are returned in Scilab workspace and can consequently be directly post-processed for statistical estimation. Figure 11 shows the post-processed resilience of a one-year simulation with several computed indicators. Figure 7 is also an output of statistics processing in Scilab.

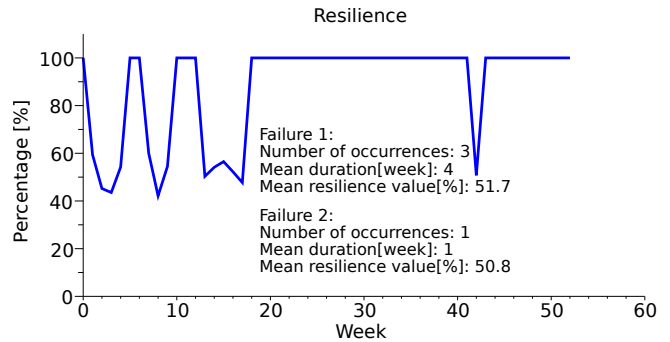


Figure 11: Resilience computed for a one-year simulation. For each type of failure, several indicators are computed like the number of occurrences or the mean duration.

## 4.3 Discussion

The presented simulation framework can be used to compare different designs of an offshore oil facility, in order to choose the most productive and/or robust, depending on the choice of a performance evaluator like those of Section 3.2.3. At the current stage of the project, the operating states and their occurring probabilities were defined arbitrarily. In further steps, they have to be set from technical data and part of the complexity of failure dependencies has to be handled automatically.

## 5 Conclusion and perspectives

We have presented our first results on the simulation of an offshore oil facility at the system level i.e., simulating all the plant components while considering also risk and failure estimations. Modelica is used to describe the physics of the flow through the various components of the offshore field. All the stochastic processes that can affect an offshore oil exploitation (failure, price variations, etc.) can be integrated in the stochastic layer of the simulation. Statistical indicators are obtained using simulation. Ongoing research is focused on up-scaling the approach and addressing industrial scale-one designs.

## Acknowledgments

The authors would like to thank Arnaud Antkowiak, associate professor at Université Pierre et Marie Curie for his help on two-phase flow modeling, Jean-Louis Grange, former senior expert at EDF and Gérard Le Coq, former head of department at EDF for their support on junction models. The authors are also

grateful to Jean-Philippe Chancelier, senior researcher at CERMICS lab of École des Ponts ParisTech, for his precious advice concerning Scilab/Scicos and its interface with Modelica. Finally, special thanks go to Lei Zhang for his several contributions to the work [17].

## References

- [1] Olaf Bauer. Modelling of Two-Phase Flows with Modelica. Master's thesis, Department of Automatic Control, Lund University, Sweden, November 1999.
- [2] Javier Bonilla, Luis J. Yebra, Sebastián Dormido, and François E. Cellier. Object-Oriented Library of Switching Moving Boundary Models for Two-phase Flow Evaporators and Condensers. In *9<sup>th</sup> International Modelica Conference*, pages 71–80, September 2012.
- [3] Daniel Bouskela, Audrey Jardin, Zakia Benjelloun-Touimi, Peter Aronsson, and Peter Fritzson. Modelling of uncertainties with Modelica. In *8<sup>th</sup> International Modelica Conference*, pages 673–685, March 2011.
- [4] Francesco Casella, Martin Otter, Katrin Proelss, Christoph Richter, and Hubertus Tummescheit. The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In *5<sup>th</sup> International Modelica Conference*, pages 631–640, September 2006.
- [5] D. Chisholm. A theoretical basis for the Lockhart-Martinelli correlation for two-phase flow. *International Journal of Heat and Mass Transfer*, 10(12):1767–1778, 1967.
- [6] Jonas Eborn and Karl Johan Åström. Modeling of a Boiler Pipe with Two-Phase Flow Instabilities. In *Modelica Workshop*, pages 79–88, October 2000.
- [7] EDF-EADS-PhiMeca. <http://www.openturns.org/>.
- [8] H. Genceli, K.A. Kuenhold, O. Shoham, and J.P. Brill. Dynamic Simulation of Slug Catcher Behavior. *SPE Annual Technical Conference and Exhibition*, October 1988.
- [9] S.E. Haaland. Simple and Explicit Formulas for the Friction Factor in Turbulent Pipe Flow. *Journal of Fluids Engineering (ASME)*, 105(1):89–90, March 1983.
- [10] Jakob Munch Jensen and Hubertus Tummescheit. Moving Boundary Models for Dynamic Simulations of Two-Phase Flows. In *2<sup>nd</sup> International Modelica Conference*, pages 235–244, 2002.
- [11] Lijun Ji, Bin Wu, Kui Chen, Jiawen Zhu, and Haifeng Liu. Momentum Correction Coefficient for Two Jet Flows Mixing in a Tee Junction. *Chemical Engineering Research and Design*, 87(8):1065–1068, 2009.
- [12] R.W. Lockhart and R.C. Martinelli. Proposed Correlation of Data for Isothermal, Two-Phase Two-Component Flow in Pipes. *Chemical Engineering Progress*, (45):39–48, 1949.
- [13] Angela Peck and Slobodan P. Simonovic. Coastal Cities at Risk (CCaR): Generic System Dynamics Simulation Models for Use with City Resilience Simulator. Technical report, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, April 2013.
- [14] Gregory Provan and Alberto Venturini. Stochastic Simulation and Inference using Modelica. In *9<sup>th</sup> International Modelica Conference*, pages 829–838, September 2012.
- [15] Scilab. <https://www.scilab.org>, ©1989-2011 (INRIA) ©1989-2007 (ENPC), Scilab Consortium (DIGITEO).
- [16] Thorben Vahlenkamp and Stefan Wischhusen. FluidDissipation for Applications - A Library for Modelling of Heat Transfer and Pressure Loss in Energy Systems. In *7<sup>th</sup> International Modelica Conference*, pages 132–141, September 2009.
- [17] Lei Zhang. Modélisation et Simulation d'une installation industrielle complexe. Technical report, Eurobios SCB/ENSTA ParisTech, 2013.



# Parameter Selection in a Combined Cycle Power Plant

Niklas Andersson<sup>a</sup> Johan Åkesson<sup>b,c</sup> Kilian Link<sup>d</sup>

Stephanie Gallardo Yances<sup>d</sup> Karin Dietl<sup>d</sup> Bernt Nilsson<sup>a</sup>

<sup>a</sup>Lund University, Department of Chemical Engineering, Lund, Sweden

<sup>b</sup>Lund University, Department of Automatic Control, Lund, Sweden

<sup>c</sup>Modelon AB, Lund, Sweden

<sup>d</sup>Siemens AG, Erlangen, Germany

## Abstract

A combined cycle power plant are modeled and considered for calibration. The dynamic model, aimed for start-up optimization, contains 64 candidate parameters for calibration. The number of parameter sets that can be created are huge and an algorithm called subset selection algorithm is used to reduce the number of parameter sets. The algorithm investigates the numerical properties of a calibration from a parameter Jacobean estimated from a simulation of the model with reasonably chosen parameter values. The calibrations were performed with a Levenberg-Marquardt algorithm considering the least squares of eight output signals. The parameter value with the best objective function value resulted in simulations in good compliance to the process dynamics. The subset selection algorithm effectively shows which parameters that are important and which parameters that can be left out.

*Keywords: Combined Cycle Power Plants; Start-up; Calibration; Parameter Selection*

## 1 Introduction

Increasing environmental awareness has resulted in more demanding requirements. Energy supply is getting more attention and more and more wind turbines and solar power stations are built to adapt to a more environmentally friendly world. The challenge however is that the sun doesn't always shine and the wind doesn't always blow. Combined cycle power plants (CCPP) work as a good complement, because of its fast startup and shutdown time. Furthermore, the CCPPs have high thermal efficiency and are relatively environmentally friendly [1].

The market energy price fluctuates every day, which

affects the profitability of the CCPPs. Because of this it is of importance to adapt to the market and be able to quickly start up and shut down the process. A quick start-up is important because energy is not produced until the gas turbine reaches full speed and is synchronized to the grid. However, this cannot be done too quickly because a rapid temperature change wears out sensible parts. The startup follows three phases: the first to accelerate the gas turbine to full speed, the second to increase the load of the gas turbine and the third to drive the steam to the steam turbine. The model of this work focuses on the second part which is the most critical during a startup. Sensible parts as the drum after the evaporator and the header of the superheater are important to model for a successful startup optimization. A model was therefore set up aimed for optimization of the startup, considering the temperature gradient in the sensible parts to estimate the tensions. The model has previously been used in optimizations [2, 3]. The startup of combined cycle power plants has been optimized in several studies before [4, 5, 6, 7, 8].

Accurate modelling of the CCPP is a difficult task, which if successful can cut expenses. This requires a good calibration of the model to make the discrepancy to the real process dynamics as small as possible. The main purpose of the calibration is to enable a valid model for optimization of startups. The parameter estimation is performed using a Levenberg-Marquardt algorithm, which effectively uses the parameter Jacobean to find the optimum.

A model usually contains many parameter candidates for calibrations. Estimating many parameter simultaneously leads to ill-conditioned calibration problem due to dependency between the parameters, that make convergence bad and with wide confidence intervals as a result. A parameter-selection algorithm,

called subset selection algorithm (SSA) is proposed that ranks the parameter by two properties,  $\alpha$  and  $\kappa$  [9]. The parameter  $\alpha$  is correlated to the size of the confidence regions for a parameter set and  $\kappa$  is a measure of how well-conditioned the parameter Jacobean for a parameter set is. The algorithm that significantly reduces the number of parameter sets to study, has recently been proven to work good for a model of a polyethylene plant [10].

## 2 Theory

### 2.1 Differential algebraic equation systems

The general non-linear index-1 differential algebraic equation (DAE) form is defined by

$$\mathbf{0} = \mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \mathbf{u}, \mathbf{p}) \quad (1)$$

$$\mathbf{y} = \mathbf{g}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \mathbf{u}, \mathbf{p}) \quad (2)$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (3)$$

where  $\mathbf{x}$ ,  $\mathbf{w}$ ,  $\mathbf{u}$  and  $\mathbf{p}$  are vectors denoting state and algebraic variables that describe the inputs and parameters of the model. In Eq 2, the output variables of the system that is subject to calibration are denoted  $\mathbf{y}$ . The initial state is defined by  $\mathbf{x}_0$  and is expressed in Eq 3. To solve the steady-state problem the state derivatives  $\dot{\mathbf{x}}$  are set to 0 and  $\mathbf{x}$  and  $\mathbf{w}$  are solved to fulfill Eq 1.

### 2.2 Non-linear regression methods for differential algebraic models

Regression methods are roughly classified into two broad categories: *gradient methods* and *direct-search methods* [11]. The former depends on accurate parameter gradients, while the latter does not. Gradient methods include the Gauss-Newton Method, the steepest descent method and the Levenberg-Marquardt method, while direct-search include the simplex method, differential evolution algorithms and pattern search [12, 13]. The Gauss-Newton method gives the best results when gradients are available [14].

The Levenberg-Marquardt method is a more stable variant of the Gauss-Newton method [12] and can be used to solve the problems of estimating dynamic parameters, where the Newton step  $\Delta \mathbf{p}$  is calculated from

$$(\mathbf{J}^T \mathbf{W} \mathbf{J} + \delta \text{diag}(\mathbf{J}^T \mathbf{W} \mathbf{J})) \Delta \mathbf{p} = \mathbf{J}^T \mathbf{W} (\hat{\mathbf{y}} - \mathbf{y}) \quad (4)$$

where  $\hat{\mathbf{y}}$  is the measurements for the output and  $\delta$  is a Levenberg-Marquardt parameter, which controls the allowed step length and is updated in each iteration,

based on the quality of the Newton step. The sensitivity matrix is defined as the parameter Jacobean matrix

$$\mathbf{J} = \frac{d\mathbf{y}}{d\mathbf{p}} \quad (5)$$

and was estimated with finite differences using the central approximation.

A single shooting approach is common to solve problems of estimating dynamic parameters [15]. It starts with a guess of the parameters. The dynamic model is then simulated, and the parameters are updated iteratively by a regression method, such as Levenberg-Marquardt method.

### 2.3 Statistics

Confidence regions are calculated to assess the quality of the parameter estimates. A confidence interval of  $1 - \beta$ , with  $n_y$  number of measurements and  $n_p$  number of parameters, is estimated from

$$\mathbf{p}^* \pm s_p T_{inv}(\beta/2, n_y - n_p) \quad (6)$$

and means that there is a probability of  $1 - \beta$  that the true parameter lies within the estimated interval. Here,  $\mathbf{p}^*$  is the calibrated parameters,  $T_{inv}$  is the inverse of the Student's T test and  $s_p$  is the estimated standard error of the parameters defined as

$$s_{p_k} = \sqrt{(\chi)_{kk}}, \quad \text{for } k = 1, \dots, n_p \quad (7)$$

where  $\chi$  is the covariance matrix calculated as

$$\chi = \sigma^2 (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1}. \quad (8)$$

where  $\sigma^2$  is the error variance and can be estimated by

$$\sigma^2 = \frac{1}{n_y - n_p} |\hat{\mathbf{y}} - \mathbf{y}|^2 \quad (9)$$

and the diagonal weighting matrix,  $\mathbf{W}$ , used to scale the outputs, is defined as

$$\mathbf{W} = \begin{pmatrix} \hat{y}_1^{-2} & 0 & \dots & 0 \\ 0 & \hat{y}_2^{-2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \hat{y}_{n_y}^{-2} \end{pmatrix} \quad (10)$$

### 2.4 Subset selection algorithm

The numerical properties of a parameter estimation can be estimated from the sensitivity matrix,  $\mathbf{J}$ . An algorithm is suggested [9] that investigates parameter

sets from a nominal operating point, and ranks the parameter sets according to two quantities: the condition number,  $\kappa$ , and the parameter selection score,  $\alpha$ . Here,  $\kappa$  is defined as the ratio between the largest and smallest singular value of  $\mathbf{J}$ , and  $\alpha$  is defined by  $\alpha(\mathbf{p}) = |\mathbf{v}|$ , where  $\mathbf{v}$  is the scaled standard error for the parameters

$$v_i = s_{pi}/p_i \quad \text{for } i = 1, \dots, n_p \quad (11)$$

The quantities  $\kappa$  and  $\alpha$  are used to estimate the parameter dependencies and the uncertainties in the parameters. A low value of  $\alpha$  shows that the estimated parameters have been accurately determined, while a low value of  $\kappa$  shows that the calibration problem are well conditioned. Both  $\alpha$  and the confidence interval are calculated from the standard error of the parameters and the quantities are closely related. A low value of  $\alpha$  indicates narrow confidence intervals. The parameter  $\kappa$  is a measure of how well the calibration problem is conditioned. Low values are preferable because it indicates that the parameters are independent of each other, while high values indicates a difficult calibration, where an inverse of the sensitivity matrix does not exist or can only be calculated with low accuracy.

Both  $\alpha$  and  $\kappa$  can be calculated without any calibration of the model, by setting some reasonable parameter values, referred to as nominal values, and determine the sensitivity matrix from a simulation. From the full sensitivity matrix,  $\alpha$  and  $\kappa$  can be calculated for every parameter subset. The sensitivity matrix for a parameter subset is created by taking the corresponding columns from the full sensitivity matrix.

## 3 Methods

### 3.1 Modelling Languages and Tools

The mathematical model has been implemented in Modelica [16], which is a high-level language for describing complex physical systems, supporting object-oriented concepts such as classes, components and inheritance. It can also encode textbook-style declarative equations. This modelling paradigm has significant advantages over the block-based paradigm that is often used in the context of physical modelling. In particular, acausal modelling systems do not require the user to solve the derivatives of a mathematical model. Instead, differential and algebraic equations may be mixed, which then typically results in a differential algebraic equation.

The calibrations in this paper have been performed using JModelica.org, which is a Modelica-based open-source platform targeted at dynamic optimization

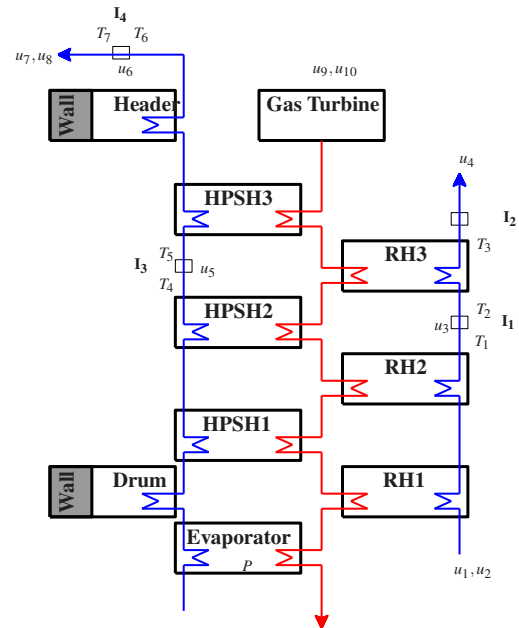


Figure 1: A schematic figure of the process.

[17]. The optimization is enabled by an extension to Modelica, Optimica, which strengthens its optimization capabilities by adding a small number of constructs. JModelica.org uses an interior point algorithm, IPOPT, to solve for feasible solutions, that fulfil the equation system[18]. Further, JModelica.org uses the Assimulo package [19], which interfaces the IDA solver from the Sundials suite [20].

To handle the high number of calibrations in this work in a reasonable time, a simple parallelization was performed. A simulation can only utilize one processor core, while it takes several simulations in every iteration of a calibration. By distributing the simulations with the python package subprocess, all eight processor cores could be utilized.

### 3.2 Mathematical plant model

A simple model scheme is found in Figure 1. The model has previously been used in startup optimization where it is described in more detail[2, 3]. The model, consisting of both differential and algebraic equations, has been derived from a combination of first principles and semi-empirical relations. It is focused on the heat recovery steam generator (HRSG) where the water side, indicated with blue arrows, are modeled by dynamic balance equations. The heat from the gas turbine (GT), shown with a red arrow, is statically modeled from the temperature ( $u_9$ ) and the mass flow ( $u_{10}$ ) of the GT. The water side is modeled as two different streams, one through the high pressure super-

Notation	Description
$u_1$	RH inlet enthalpy
$u_2$	IP mass flow
$u_3$	water injection flow $I_1$
$u_4$	IP back pressure
$u_5$	water injection $I_3$
$u_6$	water injection $I_4$
$u_7$	HP back pressure
$u_8$	HP mass flow
$u_9$	temperature GT
$u_{10}$	mass flow GT

Table 1: A list of the inputs used.

heaters (HPSHs) and the other through the reheaters (RHs). The water is evaporated in the evaporator and going through the drum before it is superheated in three steps, HPSH1, HPSH2 and HPSH3. Finally the steam is led through the header and continues to the high pressure steam turbine. The drum and the header are similarly modeled as a volume, where the wall are subject to high stress during transients that needs to be constrained. The wall is spatially discretized so that the temperature gradient can be modeled, which is an indicator of the stress. The right blue line is going through the three reheaters RH1, RH2 and RH3 and continues to the intermediate pressure steam turbine. There are also four water injections modeled, shown in boxes in the Figure, where the first  $I_1$  is located between RH2 and RH3 and  $I_{2-4}$  are located after RH3, between HPSH2 and HPSH3 and after the header, respectively. In the model there are also several valves to control the flow rates. The temperature sensors are also modeled to account for sensor lags.

The model is simulated with ten inputs following measurement data and are shown in Table 1. Three of the inputs are mass flows of the water injections, two describe the temperature and mass flow of the exhaust gas from the GT and five describe the state of the water on the HPSH and RH side. The mass flow of the exhaust gas from the GT are not measured directly, but calculated from balance equations. Eight objective signals are considered in the calibration and are shown in Table 2. The input and objective signals are also shown in Figure 1.

There are 64 potential parameters to calibrate in the model. The parameters are roughly divided in eight categories, see Table 3. Heat transfer coefficients are denoted as  $k_{in}$ , describing the transfer between the exhaust gas and metal wall,  $k_{out}$ , describing the heat transfer between the metal wall and the cold water, and  $k$ , describing either the heat transfer in the sensors or the heat transfer in the metal walls of the header

Notation	Description
$T_1$	temperature before $I_1$
$T_2$	temperature after $I_1$
$T_3$	temperature before $I_2$
$T_4$	temperature before $I_3$
$T_5$	temperature after $I_3$
$T_6$	temperature before $I_4$
$T_7$	temperature after $I_4$
$P$	pressure evaporator

Table 2: A list of the objective signals used.

	$k$	$k_{in}$	$k_{out}$	$m_{H_2O}$	$m_{Fe}$	$V$	cap	$k_v$
Header	1				7	15		
Evaporator		3	17	2	20			
Drum	10				8	4		
SH		<b>9</b>	<b>5</b>	<b>18</b>	<b>21</b>			
SH1		28	22	56	62			
SH2		29	23	57	63			
SH3		30	24	58	64			
RH		<b>12</b>	<b>6</b>	<b>19</b>	<b>11</b>			
RH1		34	25	59	31			
RH2		35	26	60	32			
RH3		36	27	61	33			
valves								13,37,38, 39,40,41
sensors	14,42,43, 44,45,46, 47,48						16,49,50, 51,52,53, 54,55	

Table 3: The parameter used in the SSA analysis. Merged parameters are indicated in bold.

and drum. There are two categories of masses, denoted as  $m_{H_2O}$  for water volumes and  $m_{Fe}$  for iron walls. The last categories are the fluid volume  $V$  for the header and drum and the heat capacity of the sensors,  $cap$ . Last category is  $k_v$  that affects the dynamics of the valves, that is modeled with a constant pressure drop. There are seven sensors measuring the outputs  $T_{1-7}$ , each with two parameters and paired as  $\{42, 49\}$ ,  $\{44, 51\}$ ,  $\{43, 50\}$ ,  $\{45, 52\}$ ,  $\{47, 54\}$ ,  $\{48, 55\}$  and  $\{46, 53\}$ .

Some parameters describe the same kind of parameter in different places of the model. For example, in the superheaters SH1, SH2 and SH3, the  $k_{out}$  parameter is described with the parameters 22, 23, 24 that has the same nominal values. A merged parameter is introduced that enables a reduction of parameters, which is important in calibration problems. For  $k_{out}$  in the superheaters, the parameter 5 is a merged parameter. Setting this parameter means that the children parameters 22, 23 and 24 gets the set value. There are 11 merged parameters in the analysis, where  $k_{in}$ ,  $k_{out}$ ,  $m_{H_2O}$  and  $m_{Fe}$  are set by the parameters 9, 5, 18, 21 for the superheaters and the parameters 12, 6, 19, 11 for the reheaters, the parameter 13 sets all the other  $k_v$  parameters and  $k$  and  $cap$  for the temperature sensors are set with the parameters 14 and 16. A merged parameter can not be in the same parameter set as its children.

Naturally, many of the parameters are highly correlated, such as  $k_{in}$  and  $k_{out}$ . For convenience the parameter set  $\{p_1, p_2, p_3\}$  is denoted as  $p_{1,2,3}$

## 4 Calibration methodology for large-scale systems

### 4.1 Calibration procedure

The calibration is made by minimising the objective function described by a least square formulation of the error between the plant data and the model response. If all parameters are included in the calibrations, it leads to badly conditioned problems. The number of parameter sets that can be combined grows rapidly with the number of parameters. To reduce the number of parameters to estimate in the model a parameter selection algorithm called subset selection algorithm was used. Information from the sensitivity matrix is used to avoid ill-conditioned parameter estimations and to find parameter sets that can be determined with low parameter uncertainty.

The calibration procedure is solved by a single shooting procedure, where the model is simulated for every iteration of parameters. An attempt is made in each iteration to find the initial states for the updated parameters. The system simulation then proceeds during the whole start-up. The initial states are found by solving a steady-state problem, defined in Eqs. (1)-(3) and with  $\dot{\mathbf{x}} = 0$ . The system is subsequently simulated, with the inputs  $\mathbf{u}$  following the measurement data. The parameters can be updated by minimising the objective value defined as the weighted sum of squares of the residuals

$$Q(\mathbf{p}) = \sum_{i=1}^{n_t} (\hat{\mathbf{y}}_i - \mathbf{y}(t_i, \mathbf{p}))^T \mathbf{W} (\hat{\mathbf{y}}_i - \mathbf{y}(t_i, \mathbf{p})), \quad (12)$$

where  $n_t$  is the number of time points and  $\mathbf{y}_z(t_i, \mathbf{p})$  is the model outputs from the simulation at time  $t_i$ . The calibration problem is solved iteratively by updating the parameters using the Levenberg-Marquardt algorithm, described in Section 2.2. The dynamic calibration is formulated as an optimization problem

$$\begin{aligned} \min_{\mathbf{p}} Q(\mathbf{p}) \\ \text{subject to Eqs. (1) - (3)} \\ \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max} \\ \mathbf{w}_{min} \leq \mathbf{w} \leq \mathbf{w}_{max} \\ \mathbf{p}_{min} \leq \mathbf{p} \leq \mathbf{p}_{max} \end{aligned} \quad (13)$$

In this work, only one data set was used for calibration and no validation was performed. In a previous work, several calibration and validation data sets were used with a similar approach on another model, which showed good compliance [10].

### 4.2 Reduction of parameter sets

Models often have many potential parameters to calibrate, where many of the parameters are dependent of each other. If all parameters are included in the calibration it results in parameter Jacobians that are highly ill-conditioned and a calibration that is impossible to solve. It is desirable to choose a subset of the parameters that are independent of each other, minimizes the objective function and with parameters that can be determined with good accuracy. The number of parameter combinations increase rapidly with the number of parameters and the exploration of all parameter sets is heavy computationally. An approach to reduce the parameter sets is suggested, where the  $\kappa$  and  $\alpha$  numbers of the SSA algorithm are used to rank the parameters. The selection method consists of two loops, the SSA loop and the calibration loop, each one consisting of three base parts: combination, evaluation and filter blocks, Figure 2. The blocks are defined as follows:

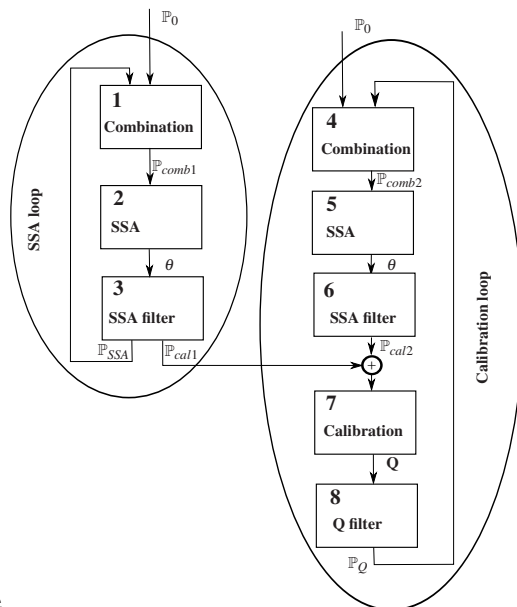
**Combination** is the process of taking an input population  $\mathbb{P}_{in}$  that contains the  $n_{pin}$  parameter sets  $\{\mathbf{p}_{in}^1, \dots, \mathbf{p}_{in}^{n_{pin}}\}$  and mixing it with all  $n_{p0}$  parameters  $\mathbb{P}_0 = \{p_1, \dots, p_{n_{p0}}\}$  to create a new parameter set population  $\mathbb{P}_{out}$  that contains parameter sets with one more parameter than the parameter sets of the input population. The input population is empty before the first iteration, and thus the output population will contain one parameter set for every parameter in  $\mathbb{P}_0$ .  $\mathbb{P}_{in}$  is not empty before the next iteration, and thus the  $\mathbf{p}_{in}^1$  will enter the output population as  $n_{p0}$  parameter sets defined by  $\{\{\mathbf{p}_{in}^1, p_1\}, \dots, \{\mathbf{p}_{in}^1, p_{n_{p0}}\}\}$ , and the parameter sets  $\mathbf{p}_{in}^2, \dots, \mathbf{p}_{in}^{n_{pin}}$  will be combined in the same way. The same parameter set can be created from two different parameter sets in the input population, and thus an operation is carried out to remove all duplicates. The maximum number of parameter sets in the output population is  $n_{p0}n_{pin}$ , but this may be reduced when duplicates are removed. There are two combination blocks, Block 1 (SSA loop), where  $\mathbb{P}_{comb1}$  is created and Block 4 (calibration loop), where  $\mathbb{P}_{comb2}$  is created.

**SSA Evaluation** evaluates  $\alpha$  and  $\kappa$  values for each

parameter set of the input population as defined in Section 2.4, and calculates a SSA score  $\theta$ , given by  $\theta = \lg \alpha + \lg \kappa$ , that is later used in the filter block to determine the best parameter sets in the SSA loop.

**Calibration** is the step where calibrations are made for all parameter sets in the input population, that consists of both  $\mathbb{P}_{cal1}$  and  $\mathbb{P}_{cal2}$ . All parameter sets are calibrated and the objective value that measures the deviation between model and measurements is returned. The calibration step is the most computationally expensive step.

**Filters** are used to reduce the number of parameter sets, which otherwise would increase rapidly. There are three filter blocks, one in the SSA loop and two in the calibration loop. The filter block takes a population of parameter sets, a score that has been calculated to rank the parameter sets, and a cutoff that defines how many parameter sets should pass. In Block 2 and 5,  $\theta$  is used as score and  $n_{SSA1}$  is used as cutoff for  $\mathbb{P}_{SSA}$ ,  $n_{cal1}$  for  $\mathbb{P}_{cal1}$  and  $n_{cal2}$  for  $\mathbb{P}_{cal2}$ . In Block 8,  $Q$  is used as score and  $n_Q$  is used as cutoff. The choice of the cutoffs are arbitrarily, but should not be chosen too small for a good analysis.



The

Figure 2: The SSA selection procedure used.

SSA evaluation is relatively cheap, but the number of parameter sets increase rapidly as  $n_p$  increases. The number of parameter sets increases as the binomial coefficients  $\binom{n_{P0}}{n_p}$ , which for  $n_{P0} = 64$  are

$\{64, 2016, 41664, 635376, 7624512, \dots\}$ . Setting a filter cutoff limits the population that must be examined to  $n_{SSA}n_{P0}$  per iteration instead. The number of calibrations are dependent of  $n_{P0}$  and the filters  $n_{cal1}$  and  $n_{cal2}$ . In the first iteration,  $n_{cal1}$  calibrations were performed and in the following rounds,  $n_{cal1} + n_{cal2}$  calibrations are performed.

In this work, the cutoffs have been chosen to  $n_{SSA} = 300$ ,  $n_{cal1} = 5$  (10 in the first iteration),  $n_{cal2} = 4$  and  $n_Q = 1$  in the work presented here. In this work the loops were iterated for parameter sets ranging from one parameter to seven parameters. The total number of calibrations performed is around five in the first iteration and nine in the rest, totally 59 calibrations.

## 5 Calibration results

### 5.1 Calibration

The calibration of the model was done for many parameter sets in the calibration loop of the SSA method. The parameter set with the best objective value with seven parameters,  $p_{6,13,16,17,22,24,47}$ , is called  $C_3$  and is shown in Figure 3. The parameters in the set consist of four  $k_{out}$  parameters for SH1, SH3 and all RHs, one valve parameter and two sensor parameters. No parameters for masses, volumes and heat transfer in walls and gas side are represented. The calibration reduces the objective function value from 1.85 with nominal parameter values to 0.585 and improves all eight objective signals. The largest improvements are for  $T_1$ ,  $T_4, T_6$  and  $P$ , with 72%, 70%, 87% and 74% reduction.

In Table 4 the calibrated parameters with confidence interval for the three calibrations  $C_{1-3}$  are compared. The confidence intervals are narrow for all parameters, except for  $p_{24}$  that is  $k_{out}$  in SH3. This parameter affects mainly the objective  $T_6$ , which transient for the nominal parameter value are far below the measurements. To increase the temperature, the obtained parameter value is therefore very high, with the values 64.9, 79.3 and 79.5. The confidence intervals are almost as big as the parameter which is a very inaccurate parameter. The  $\alpha$  and  $\kappa$  values are also notably worse at the optimum.

### 5.2 Parameter selection

The SSA analysis was performed for the parameters and is shown in Figure 4. The evaluated parameter sets results in dot clouds that move upwards and to the right, the more parameters that are added. The dot

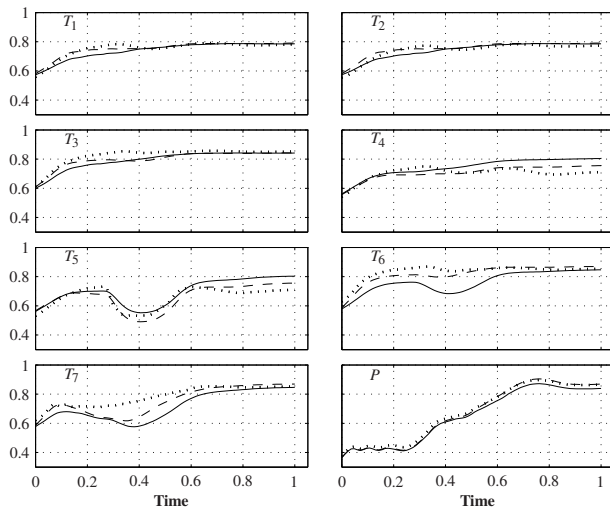


Figure 3: Simulation profiles from the best calibration for all states in the objective. The measurement data (dotted line) are shown together with the simulation with nominal (solid line) and optimal (dashed line) parameter values are shown in solid and dashed line.

parameter	$C_1$	$C_2$	$C_3$
$p_6$	$3.89 \pm 0.45$	$4.28 \pm 0.62$	$4.58 \pm 0.71$
$p_{13}$	$1.97 \pm 0.027$	$2.17 \pm 0.043$	$2.16 \pm 0.042$
$p_{16}$		$0.92 \pm 0.023$	$0.964 \pm 0.027$
$p_{17}$		$0.66 \pm 0.020$	$0.665 \pm 0.020$
$p_{22}$		$0.38 \pm 0.014$	$0.375 \pm 0.014$
$p_{23}$	$0.54 \pm 0.021$		
$p_{24}$	$64.9 \pm 46.2$	$79.3 \pm 78.7$	$79.5 \pm 79.5$
$p_{47}$	$9.6 \pm 0.078$		$1.26 \pm 0.069$

Table 4: Calibrated parameter values with a 95% confidence interval for calibrations  $C_1$  and  $C_2$ . All parameters are scaled with the nominal parameter value.

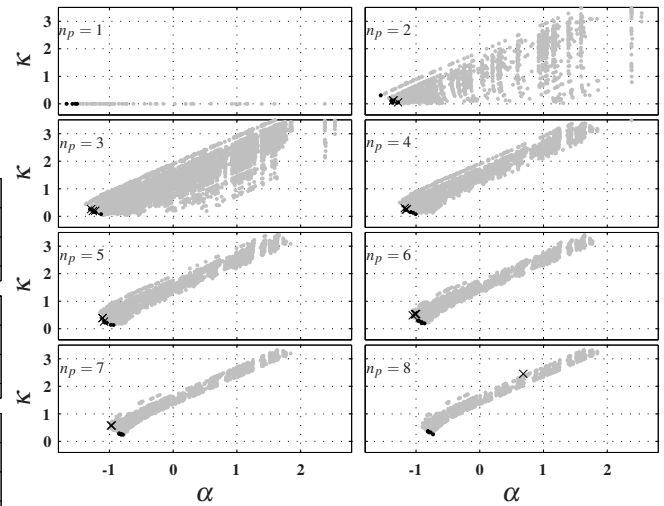


Figure 4: The SSA analysis for  $n_p$  from 1 to 8. The dot clouds move to the right and up when parameters are added. Parameter sets with best  $\theta$  is marked as (•) and parameters from the best objective loop (×)

clouds are created from the SSA loop of the SSA analysis, with 64 parameter sets for  $n_p = 1$  and  $64 \cdot 300$  parameter sets for  $n_p$  from two to eight. The parameter sets with the lowest values of  $\theta$  ( $\mathbb{P}_{cal1}$ ) are shown with black dots and are located in the lower left corner in each figure. Those parameter sets were calibrated and the objective values are shown in Table 5. The objective value improves when parameters are added for  $n_p$  from one to five. After that the calibration gets worse for  $n_p$  equal to six and seven.

In the calibration loop of the SSA analysis, the best parameter sets were combined with new parameters to create  $\mathbb{P}_{cal2}$  that were calibrated and is presented in Table 6. Those parameter sets, marked with crosses are also located in the lower left corner for  $n_p$  from two to seven. The objective values of this loop gets better for every iteration as new parameters are added to the best parameter set of the previous iteration until eight parameters are reached where the calibrations do not converge. Also the  $\alpha$  and  $\kappa$  values for those parameter sets are very bad.

In Table 5 there is 20 unique parameters  $\{5, 6, 12, 13, 14, 16, 17, 22, 23, 24, 26, 43, 45, 46, 47, 48, 52, 53, 54, 55\}$  where  $\{5, 6, 12, 13, 14, 16\}$  are merged parameters. Seven of the parameters,  $\{5, 6, 17, 22, 23, 24, 26\}$  are  $k_{out}$  parameters, while only one parameter,  $\{12\}$  is a  $k_{in}$  parameter, indicating that the parameters for heat transfer between the metal wall to the cold water have greater impact than the heat transfer between the exhaust gas and the metal wall. No drum or header parameter are visible in the best parameter

Table 5: The calibration results from the left loop of the SSA analysis, sorted by  $\theta$ . \* $C_1$  † The 8<sup>th</sup> value of  $\theta$

parameters	$\log_{10}(\alpha)$	$\log_{10}(\kappa)$	$\theta$	obj
$p_5$	-1.87	0	-1.87	1.77
$p_{14}$	-1.65	0	-1.65	1.77
$p_{16}$	-1.65	0	-1.65	1.77
$p_{17}$	-1.56	0	-1.56	1.85
$p_{22}$	-1.55	0	-1.55	1.85
† $p_{24}$	-1.46	0	-1.46	1.03
<hr/>				
$p_{6,23}$	-1.51	0.0369	-1.48	1.79
$p_{6,22}$	-1.54	0.0947	-1.45	1.79
$p_{6,17}$	-1.54	0.109	-1.43	1.8
$p_{5,14}$	-1.7	0.31	-1.39	1.73
$p_{5,16}$	-1.7	0.31	-1.39	1.73
<hr/>				
$p_{6,23,24}$	-1.67	0.152	-1.52	0.832
$p_{6,17,24}$	-1.69	0.19	-1.5	0.895
$p_{6,24,47}$	-1.64	0.147	-1.49	0.899
$p_{6,24,54}$	-1.64	0.148	-1.49	0.899
$p_{6,22,24}$	-1.68	0.192	-1.49	0.808
<hr/>				
$p_{6,17,24,47}$	-1.6	0.225	-1.38	0.895
$p_{6,17,24,54}$	-1.6	0.225	-1.38	0.895
$p_{13,26,46,48}$	-1.44	0.0756	-1.36	1.53
$p_{13,26,46,55}$	-1.44	0.0756	-1.36	1.58
$p_{13,26,48,53}$	-1.44	0.0761	-1.36	1.53
<hr/>				
* $p_{6,13,23,24,47}$	-1.5	0.227	-1.28	0.675
$p_{6,13,23,24,54}$	-1.5	0.227	-1.28	0.675
$p_{12,13,46,48,54}$	-1.41	0.144	-1.27	1.53
$p_{12,13,46,54,55}$	-1.41	0.144	-1.27	1.58
$p_{12,13,46,47,48}$	-1.41	0.144	-1.27	1.53
<hr/>				
$p_{12,13,45,46,48,54}$	-1.34	0.202	-1.14	1.52
$p_{12,13,45,46,54,55}$	-1.34	0.202	-1.14	1.57
$p_{12,13,45,48,53,54}$	-1.34	0.202	-1.14	1.52
$p_{12,13,45,53,54,55}$	-1.34	0.202	-1.14	1.56
$p_{12,13,45,46,47,48}$	-1.34	0.203	-1.14	1.52
<hr/>				
$p_{13,26,43,45,46,48,54}$	-1.27	0.264	-1.01	1.49
$p_{13,26,43,45,46,54,55}$	-1.27	0.264	-1.01	1.54
$p_{13,26,43,45,48,53,54}$	-1.27	0.264	-1.01	1.49
$p_{13,26,43,45,53,54,55}$	-1.27	0.264	-1.01	1.54
$p_{13,26,43,46,48,52,54}$	-1.27	0.264	-1.01	1.5

sets of the analysis. For the valves, only the merged parameter {13} is visible in the tables. There are many sensor parameters visible in the result, namely {43,45,46,47,48,52,53,54,55}, corresponding to the sensors for  $T_{1,4,5,6,7}$ .

In Table 6 there is only 11 unique parameters 6, 13, 14, 16, 17, 22, 24, 47, 48, 54, 55. All of those are not surprisingly also visible in Table 5, because they are partly derived from the best parameter sets of that table.

The parameter set with the lowest objective value for parameter sets with one parameter is  $p_{24}$  with  $Q = 1.03$ . This parameter is also visible for all of the best parameter sets, even though the confidence intervals are wide. The  $\alpha$  and  $\kappa$  values at the optimum were much worse at the optimum than for the nominal parameter values.

The parameter sets with  $p_{47}$  and  $p_{54}$  are replaceable in several places, for instance in  $p_{6,22,24,47}$  and  $p_{6,22,24,54}$  that give the same objective value. Both of

Table 6: The calibration results from the right loop of the SSA analysis, sorted by  $\theta$ . \*\* $C_2$  \*\*\* $C_3$

parameters	$\log_{10}(\alpha)$	$\log_{10}(\kappa)$	$\theta$	obj
$p_{17,24}$	-1.36	0.119	-1.24	1.04
$p_{24,47}$	-1.28	0.0518	-1.23	1.04
$p_{24,54}$	-1.28	0.0522	-1.23	1.04
$p_{22,24}$	-1.36	0.125	-1.23	1
<hr/>				
$p_{6,22,24}$	-1.26	0.192	-1.07	0.808
$p_{16,22,24}$	-1.29	0.263	-1.02	0.935
$p_{14,22,24}$	-1.29	0.263	-1.02	0.935
$p_{22,24,47}$	-1.22	0.204	-1.02	1
<hr/>				
$p_{6,22,24,47}$	-1.16	0.24	-0.923	0.806
$p_{6,22,24,54}$	-1.16	0.24	-0.923	0.806
$p_{6,13,22,24}$	-1.15	0.256	-0.89	0.662
$p_{6,17,22,24}$	-1.18	0.299	-0.88	0.773
<hr/>				
$p_{6,13,22,24,47}$	-1.08	0.262	-0.823	0.659
$p_{6,13,22,24,54}$	-1.08	0.262	-0.823	0.663
$p_{6,13,16,22,24}$	-1.11	0.389	-0.718	0.655
$p_{6,13,14,22,24}$	-1.11	0.39	-0.717	0.741
<hr/>				
** $p_{6,13,16,17,22,24}$	-1.05	0.488	-0.563	0.583
$p_{6,13,16,22,24,47}$	-1.02	0.528	-0.487	0.655
$p_{6,13,16,22,24,54}$	-1.01	0.528	-0.487	0.655
$p_{6,13,16,22,24,48}$	-0.999	0.536	-0.463	0.666
<hr/>				
*** $p_{6,13,16,17,22,24,47}$	-0.975	0.576	-0.398	0.585
$p_{6,13,16,17,22,24,54}$	-0.975	0.576	-0.398	0.585
$p_{6,13,16,17,22,24,48}$	-0.961	0.579	-0.383	0.598
$p_{6,13,16,17,22,24,55}$	-0.961	0.579	-0.382	0.596

those are sensor parameters for output  $T_5$ . Also the  $T_6$  parameters  $p_{48}$  and  $p_{55}$  seem replaceable, apart from some small difference in objective value.

## 6 Discussion and summary

The objective function values became better when adding more parameters, but reached a point where adding of parameters made the calibrations too hard to solve. The best parameter set ( $C_3$ ) chose parameters from different parts of the model to minimize as many outputs as possible in the objective function.

The trend in the calibrations is that the parameter sets get harder to solve when more parameters are added and take more iterations. For eight parameters and more the calibrations fail to converge more often. Apart from that the calibrations are more complex when parameters are added, it is also harder to find independent parameters for a larger parameter set. Ill-conditioned calibration problems leads to parameter steps that make the simulations infeasible.

The analysis shows that the  $k_{out}$  parameters occur more frequently than the  $k_{in}$  parameters and indicates that the cold water side has greater impact of the model than the exhaust gas side. This is probably because the exhaust gas is only statically modeled in contrast to the cold water side. The analysis also shows that some parameters are replaceable, such as  $p_{47}$  and  $p_{54}$ . Only



one of the parameters are therefore needed for further analysis.

The merged parameters performed well in the analysis where six of the 11 merged parameters appeared in the best parameter sets. Merged parameters are effective for parameters that is expected to behave similarly and keeps the number of parameters in the calibration problem less.

The information to the SSA analysis is estimated from uncalibrated parameters but give a good indication about the best parameter sets considering the  $\alpha$  and  $\kappa$  values. The values are dependent of the parameter values and will obviously change for the calibrated parameter values, but hopefully not much. For most parameter sets, the  $\alpha$  and  $\kappa$  values stayed roughly the same, but for parameter sets including  $p_{24}$  resulted in worse  $\alpha$  and  $\kappa$  values at the optimum. Still, the analysis highlights  $p_{24}$  as a crucial parameter, that can decrease the objective function value the most, but with very wide confidence intervals as a result. A further analysis is required to understand this behavior.

Both the SSA and calibration loop of the analysis is dependent of the cutoff numbers for good performance. The calibration numbers were consciously set to low numbers, because of long calibration times that were performed on a single computer. The numbers can be set higher for a more thorough analysis if time or a computer cluster is available. The result shown here proves that satisfactory calibration results can be reached even with low cutoff numbers.

The parameter estimation results are in good compliance to the process dynamics. The subset selection algorithm effectively shows which parameters that are important and which parameters that can be left out. Considering the few number of calibrations that were performed, the result is satisfactory.

## References

- [1] Kehlhofer, R., Warner, J., Nielsen, H., Backmann, R., Combined-Cycle Gas and Steam Turbine Power Plants. ISBN: 0-87814-736-5, second edition, PennWell Publishing Company, Tulsa, Oklahoma, USA, 1999.
- [2] Lind, A., Sällberg, E., Velut, S., Åkesson, J., Gallardo Yances, S., Link, K. Sep. 2012. Start-up Optimization of a Combined Cycle Power Plant, 9th International Modelica Conference. Munich, Germany.
- [3] Lind, A., Sällberg, E. Optimization of the Start-up Procedure of a Combined Cycle Power Plant, Master's Thesis, Lund University, Department of Automatic Control, 2012
- [4] Casella, F., Pretolani, F. Fast Start-up of a Combined-Cycle Power Plant: A Simulation Study with Modelica. In: Modelica Conference pp. 3–10, Vienna, Austria, 2006.
- [5] Casella, F. Leva, A. Modelica open library for power plant simulation: design and experimental validation. In: Proceedings of 3rd International Modelica Conference, pp. 41–50. Linköping, Sweden, 2003.
- [6] Casella, F., Farina, M., Righetti, F., Scattolini, R., Faille, D., Davelaar, F., Tica, A., Gueguen, H. Dumur, D. An optimization procedure of the start-up of combined cycle power plants. In: 18th IFAC World Congress, pp. 7043–7048. Milano, Italy, 2011.
- [7] Casella, F., Donida, F., Åkesson, J. Object-oriented modeling and optimal control: a case study in power plant start-up. In: 18th IFAC World Congress, pp. 9549–9554. Milano, Italy, 2011.
- [8] Shirakawa, M., Nakamoto, M., Hosaka, S. Dynamic simulation and optimization of start-up processes in combined cycle power plants. In: JSME International Journal, vol. 48 (1), pp. 122–128, 2005.
- [9] Cintrón-Arias, A., Banks, H. T., Capaldi, A., Lloyd, A. L., 2009. A Sensitivity Matrix Based Methodology for Inverse Problem Formulation. Journal of Inverse and Ill-Posed Problems, 17(6), 545-564.
- [10] Andersson N., Larsson, P.-O., Åkesson, J., Carlsson, N., Skålén, S. Nilsson, B. Parameter selection in the parameter estimation of grade transitions in a polyethylene plant. submitted for publication.
- [11] Edgar, R., Himmelblau, D., 1988. Optimization of Chemical Processes, 1st Edition. McGraw-Hill, New York, NY.
- [12] Englezos, P., Kalogerakis, N., 2000. Applied parameter estimation for chemical engineers, 1st Edition. CRC Press.

- [13] Storn, K., Price, R., Lampinen, J. 2005 Differential Evolution – A Practical Approach to Global Optimization, Springer-Verlag, Berlin.
- [14] Comparison of gradient methods for the solution of nonlinear parameter estimation problems. *SIAM Journal on Numerical Analysis* 7 (1), 157-186.  
URL <http://www.jstor.org/stable/2949590>
- [15] Vassiliadis, V., 1993. Computational solution of dynamic optimization problem with general differential-algebraic constraints. Ph.D. thesis, Imperial Collage, London, UK
- [16] The Modelica Association, 2011. The Modelica Association Home Page. <http://www.modelica.org>
- [17] Åkesson, J., and Årzén, K.-E., Gäfvert, M., Bergdahl, T., Tummescheit, H., nov 2010. Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for solving large-scale dynamic optimization problem. *Computers and Chemical Engineering* 34 (11), 1737–1749.
- [18] Wächter, A., Biegler, L. T., 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* 106 (1), 25–58
- [19] Andersson, C., Andreasson, J., Führer, C., Åkesson, J., 2012. A workbench for multibody systems ode and dae solvers. In: 2nd Joint International Conference on Multibody System Dynamics. Stuttgart, Germany.
- [20] Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., Woodward, C. S., September 2005. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.* 31, 363–396  
URL <http://doi.acm.org/10.1145/1089014.1089020>

# Restarting algorithms for simulation problems with discontinuities

Fatemeh Mohammadi Carmen Arévalo Claus Führer\*  
Numerical Analysis, Center of Mathematical Sciences, Lund University  
Sölvegatan 18, SE-22100 Lund, Sweden

## Abstract

Modelica's language support includes so-called events for describing discontinuities. Modern integrating environments, like Assimulo, provide elaborate event detection and event handling methods. In addition, the overall performance of a simulation of models with discontinuities (hybrid models) depends strongly on the methods for restarting the integration after an event detection. The present paper reviews two restarting methods for multistep methods, both based on Runge–Kutta starters, and presents preliminary first experiments with Assimulo and LSODAR as a proof of concept, which motivates to apply the technique to hybrid systems described in Modelica and simulated by JModelica.org/PyFMI and Assimulo [1, 3, 2].

*Keywords: events, discontinuities, hybrid systems, multistep method, Runge–Kutta method, simulation restart*

## 1 Introduction

When dealing with hybrid systems, i.e. dynamic systems with state or time discontinuities, much emphasis has been put on the modeling aspect. Attempts to standardize the formulation of events and algorithms for event detection were in the focus of development and research, e.g. [4]. On the other hand, the question of restarting complex integration methods like multistep methods, with their sophisticated internal error and order control algorithms and internal data representations, did not attract much attention. In this paper we want to take up and review two early ideas for restarting and to present some experiments using the JModelica.org - PyFMI - Assimulo toolchain.

A multistep method is classically started by stepwise increasing the order of the method, starting with a

first order method (implicit Euler method) and leading to a method having the operational order of the problem at hand. Simulations are often done for a set of parameterized models for which the operational order and also good guesses of initial step sizes are available from other simulation runs. Thus, a goal for improving the integration performance is to avoid costly starting phases and directly start the integrator with a method already having the operational order. To start such a higher order method several internal values are required. Here we consider two ideas for providing these values. In both cases the starting values are obtained from the stage values of a single Runge–Kutta step of a specially designed method. One of them uses state values, while the other is geared to Nordsieck based multistep methods like LSODAR.

## 2 Runge–Kutta starter with state values

We demonstrate the principle by constructing a Runge–Kutta starter for a third order multistep method, [8].

Furthermore, we construct two error estimates for determining the starting step of both the Runge–Kutta starter and a class of multistep methods, i.e. Adams methods.

Such a Runge–Kutta starter has to have an internal stage of order 3 as soon as possible and all subsequent stages need to be at least of order 3. In addition; the final result should be of order 4 for the purpose of error estimation. It is well-known that to get third-order accuracy at least three internal stages are necessary, and to conserve this accuracy for subsequent stages we need to aim for a Runge–Kutta method with at least six stages, [5]. We will thus consider a 6-stage Runge–Kutta method.

For the initial value problem

$$y' = f(t, y), \quad y(0) = y_0, \quad (1)$$

\*partly supported by LCCC - Lund Center for Control of Complex Engineering Systems

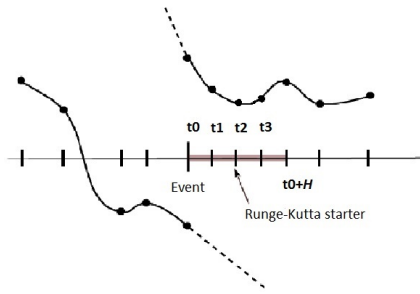


Figure 1: Runge–Kutta starter after a discontinuity

an  $s$ -stage explicit Runge–Kutta method can be written in the form,

$$\begin{aligned}
 k_i &:= f(t_0 + c_i H, g_{i-1}), \\
 g_i &:= y_0 + H \sum_{j=1}^i a_{ij} k_j, \quad i = 1, \dots, s, \\
 y_1 &:= y_0 + H \sum_{j=1}^s b_j k_j,
 \end{aligned} \tag{2}$$

where  $y_1$  is the numerical solution at  $t_1 = t_0 + H$ ,  $H$  is the Runge–Kutta step size and  $k_i$  are stage derivatives.

$f$  may be discontinuous but it is assumed to be piecewise smooth.

In the construction of an order 4, 6-stage explicit Runge–Kutta method, order conditions up to order four need to be satisfied. Let

$$\begin{aligned}
 b &:= (b_1, b_2, \dots, b_s)^T, \\
 a_i &:= (a_{i1}, a_{i2}, \dots, a_{ii}), \\
 C_i &:= \text{diag}(c_1, \dots, c_i), \\
 A_i &:= (a_{jk})_{j,k=1}^i, \\
 e_i &:= (1, 1, \dots, 1)^T.
 \end{aligned} \tag{3}$$

When deriving the stage order conditions, we make use of the fact that in Eq. (2), the stage values  $g_i$  and  $y_i$  have structurally the same form. Therefore, we can derive the order conditions for internal stages in the same way.

The order conditions for a fourth-order Runge–Kutta method are

- order 1

$$b^T e_s = 1.$$

- order 2

$$b^T C_s e_s = \frac{1}{2}.$$

- order 3

$$b^T C_s^2 e_s = \frac{1}{3},$$

$$b^T A_s C_s e_s = \frac{1}{6}.$$

- order 4

$$b^T C_s^3 e_s = \frac{1}{4},$$

$$b^T C_s A_s C_s e_s = \frac{1}{8},$$

$$b^T A_s C_s^2 e_s = \frac{1}{12},$$

$$b^T A_s^2 C_s e_s = \frac{1}{24}.$$

(4)

Additionally we require

$$\sum_{j=1}^s a_{ij} = c_i. \tag{5}$$

The remaining order conditions for internal stages  $i = 4, 5, 6$  are

- order 2

$$a_i^T C_i e_i = \frac{1}{2} c_i^2.$$

- order 3

$$a_i^T C_i^2 e_i = \frac{1}{3} c_i^3,$$

$$a_i^T A_i C_i e_i = \frac{1}{6} c_i^3.$$

(6)

We want to both obtain third-order accuracy and minimize the truncation error bound. Ralston [6] showed that the third-order Runge-Kutta method which has the minimal error bound among all third-order Runge–Kutta methods is

$$\begin{aligned}
 k_1 &= hf(t_n, y_n), \\
 k_2 &= hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1), \\
 k_3 &= hf(t_n + \frac{3}{4}h, y_n + \frac{3}{4}k_2), \\
 y_{n+1} &= y_n + \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3.
 \end{aligned} \tag{7}$$

This implies the Butcher tableau for the first four stages is

0	0		
$\frac{1}{2}c_4$	$\frac{1}{2}c_4$	0	
$\frac{3}{4}c_4$	0	$\frac{3}{4}c_4$	0
$c_4$	$\frac{2}{9}c_4$	$\frac{1}{3}c_4$	$\frac{4}{9}c_4$

From condition (5) we find

$$a_{21} = c_2, \quad a_{31} = (1 - \theta)c_3, \quad a_{32} = \theta c_3. \quad (8)$$

We now substitute (6) and (5) in (4) to calculate the values  $b_i$  for the six stage Runge–Kutta method.

$$\begin{aligned} b_2c_2^3 + b_3c_3^3 + b_4c_4^3 + b_5c_5^3 + b_6c_6^3 &= \frac{1}{4}, \\ b_3c_3\theta c_3c_2 + \frac{1}{2}b_4c_4^3 + \frac{1}{2}b_5c_5^3 + \frac{1}{2}b_6c_6^3 &= \frac{1}{8}, \\ b_3c_3\theta c_2^2 + \frac{1}{3}b_4c_4^3 + \frac{1}{3}b_5c_5^3 + \frac{1}{3}b_6c_6^3 &= \frac{1}{12}, \\ \frac{1}{6}b_4c_4^3 + \frac{1}{6}b_5c_5^3 + \frac{1}{6}b_6c_6^3 &= \frac{1}{24}. \end{aligned} \quad (9)$$

For the first three stages we require  $c_2 \neq 0, c_3 \neq 0$  and  $\theta \neq 0$ , otherwise a third-order Runge–Kutta method cannot be obtained. So  $b_2 = b_3 = 0$  and Equations (9) reduce to a single equation

$$b_4c_4^3 + b_5c_5^3 + b_6c_6^3 = \frac{1}{4}.$$

We repeat this process for order 2 and 3 conditions, getting

$$b_4c_4^2 + b_5c_5^2 + b_6c_6^2 = \frac{1}{3},$$

and

$$b_4c_4 + b_5c_5 + b_6c_6 = \frac{1}{2}.$$

respectively.

Here we have a system of equations for given  $c_4, c_5, c_6$ ,

$$\begin{pmatrix} c_4 & c_5 & c_6 \\ c_4^2 & c_5^2 & c_6^2 \\ c_4^3 & c_5^3 & c_6^3 \end{pmatrix} \begin{pmatrix} b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{3} \\ \frac{1}{2} \end{pmatrix}.$$

We obtain a Vandermonde type matrix, which has, for distinct  $c_4, c_5$  and  $c_6$ , a unique solution:

$$\begin{aligned} b_1 &= 1 - b_4 - b_5 - b_6, \\ b_4 &= \frac{3 - 4c_5 - 4c_6 + 6c_5c_6}{12c_4(c_4 - c_5)(c_4 - c_6)}, \\ b_5 &= \frac{3 - 4c_4 - 4c_6 + 6c_4c_6}{12c_5(c_4 - c_5)(c_5 - c_6)}, \\ b_6 &= \frac{3 - 4c_4 - 4c_5 + 6c_4c_5}{12c_6(c_4 - c_6)(c_5 - c_6)}. \end{aligned}$$

In order to obtain an equidistant grid for starting multistep methods, we can choose

$$c_4 = \frac{1}{4}, \quad c_5 = \frac{1}{2}, \quad c_6 = \frac{3}{4},$$

which gives

$$b_1 = b_2 = b_3 = 0, \quad b_4 = \frac{2}{3}, \quad b_5 = -\frac{1}{3}, \quad b_6 = \frac{2}{3}.$$

Finally, by solving the equations that guarantee the remaining order conditions, we obtain the Butcher tableau for the Runge–Kutta starter:

0	0					
$\frac{1}{8}$	$\frac{1}{8}$	0				
$\frac{3}{16}$	0	$\frac{3}{16}$	0			
$\frac{1}{4}$	$\frac{1}{18}$	$\frac{1}{12}$	$\frac{1}{9}$	0		
$\frac{1}{2}$	$\frac{5}{12}$	$-\frac{1}{3}$	$-\frac{4}{9}$	1	0	
$\frac{3}{4}$	$-\frac{1}{4}$	$\frac{3}{4}$	1	$-\frac{3}{2}$	$\frac{3}{4}$	0
	0	0	0	$\frac{2}{3}$	$-\frac{1}{3}$	$\frac{2}{3}$

We can apply the explicit Runge–Kutta starter to start  $k = 3$ -step Adams methods. We need  $k$  data points  $(t_i, f_i), i = n - k + 1, \dots, n$  to compute the respective polynomials for either the Adams–Moulton corrector or the Adams–Bashforth predictor.

### 3 Error estimation and step size control

The error of the numerical solution depends on the function  $f$  and on the step size  $H$ . The step size influences the size of the global error increment. Thus, for a given tolerance the step size is chosen in such a way that the global error increment meets a user-supplied tolerance bound.

We use an embedded formula to obtain an error estimate for the Runge–Kutta starter of the Adams method. The estimation can be done by reusing the available stages to produce a formula of different order. To do so, we apply stages  $k_4, k_5, k_6$  of the Runge–Kutta method in Section 2 and obtain  $\hat{y}_1$  by the third-order Adams–Bashforth method. We generate the difference table

$$\begin{aligned} c_4 = h & \quad k_4 \\ & \quad \quad \quad \nabla k_5 \\ c_5 = 2h & \quad k_5 \quad \quad \quad \nabla^2 k_6 \\ & \quad \quad \quad \nabla k_6 \\ c_6 = 3h & \quad k_6 \end{aligned}$$

where  $h = \frac{H}{4}$  and  $H$  is the Runge–Kutta step size. The third-order Adams–Bashforth method is

$$\hat{y}_1 = g_6 + h \sum_{i=1}^3 \gamma_{i-1} \nabla^{i-1} k_6 = g_6 + h \sum_{i=1}^3 \gamma_i^* k_{i+3}, \quad (10)$$

The latter is the Lagrange form of the Adams–Bashforth method and

$$\begin{aligned} \gamma_1^* &= \gamma_2 = \frac{5}{12}, \\ \gamma_2^* &= -\gamma_1 - 2\gamma_2 = -\frac{4}{3}, \\ \gamma_3^* &= \gamma_0 + \gamma_1 + \gamma_2 = \frac{23}{12}. \end{aligned}$$

As we have

$$g_6 = y_0 + H \sum_{j=1}^5 a_{6j} k_j, \quad (11)$$

we can rewrite equation (10) as

$$\hat{y}_1 = y_0 + H \sum_{j=1}^6 \hat{b}_j k_j$$

Thus, the error estimate is

$$\begin{aligned} y_1 - \hat{y}_1 &= \\ &= y_0 + H \sum_{j=1}^6 b_j k_j - \left( y_0 + H \sum_{j=1}^6 \hat{b}_j k_j \right) = \\ &= h \sum_{j=1}^6 \hat{e}_j k_j, \quad (12) \end{aligned}$$

giving the following coefficients

j	1	2	3	4	5	6
$\hat{b}_j$	$-\frac{1}{4}$	$\frac{3}{4}$	1	$-\frac{67}{48}$	$\frac{5}{12}$	$\frac{23}{48}$
$\hat{e}_j$	$\frac{1}{4}$	$-\frac{3}{4}$	-1	$\frac{99}{48}$	$-\frac{3}{4}$	$\frac{3}{16}$

This error estimation is the difference of a third-order predictor and the fourth-order result of the Runge–Kutta method that is applied to determine the step size for the Runge–Kutta starter.

We will now develop a second error estimate, to determine a step size for Adams method. We evaluate the right-hand side function  $f$  at the solution value  $y_1$  and call it  $k_7$ . Then we generate the third-order Adams–Moulton corrector using  $k_5, k_6, k_7$ ,

$$\begin{aligned} c_5 &= h & k_5 & & & & \\ & & & & \nabla k_6 & & \\ c_6 &= 2h & k_6 & & & \nabla^2 k_7 & \\ & & & & \nabla k_7 & & \\ c_7 &= 3h & k_7 & & & & \end{aligned}$$

The third-order approximation by the Adams–Moulton method is

$$\tilde{y}_1 = g_6 + h \sum_{i=1}^3 \beta_{i-1} \nabla^{i-1} k_7 = g_6 + h \sum_{i=1}^3 \beta_i^* k_{i+4}, \quad (13)$$

where the latter is the Lagrange form of the Adams–Moulton corrector and

$$\begin{aligned} \beta_1^* &= \beta_2 = -\frac{1}{12}, \\ \beta_2^* &= -\beta_1 - 2\beta_2 = \frac{2}{3}, \\ \beta_3^* &= \beta_0 + \beta_1 + \beta_2 = \frac{5}{12}. \end{aligned}$$

From Equation (11) we can rewrite the third-order corrector in Runge–Kutta form

$$\tilde{y}_1 = y_0 + H \sum_{j=1}^7 \tilde{b}_j k_j.$$

resulting in the following table:

j	1	2	3	4	5	6	7
$\tilde{b}_j$	$-\frac{1}{4}$	$\frac{3}{4}$	1	$-\frac{3}{2}$	$\frac{35}{48}$	$\frac{1}{6}$	$\frac{5}{48}$
$\tilde{e}_j$	$\frac{1}{4}$	$-\frac{3}{4}$	-1	$\frac{13}{6}$	$-\frac{51}{48}$	$\frac{1}{2}$	$-\frac{5}{48}$

The error estimate is used in determining the step size for starting the third-order Adams–Moulton method.

## 4 Runge–Kutta starter as an extrapolation method

The starting values of a multistep method can also be stored as a differentiation array, which constitutes the Nordsieck vector of scaled derivatives  $\frac{h^i y^{(i)}}{i!}$ ,  $i = 0, \dots, p$ . It is possible to convert a vector of state values at consecutive grid points into a Nordsieck array and vice versa without loss of accuracy. Classical multistep codes like LSODAR are based on Nordsieck formulations.

Based on such a Nordsieck formulation an alternative way of constructing a Runge–Kutta starter was developed by Gear, [5]. Here, the asymptotic expansion

of the global error of a base method is used to construct a Runge-Kutta method with higher order stage values.

We use the explicit Euler method as a base method to compute  $y_i^m$  (the super-script  $m$  refers to the corresponding step size,  $h_m = \frac{H}{m}$ ) for  $i = 1, \dots, m$ ,  $m = p, p-1, \dots, 1$ . From these values the terms in the asymptotic expansion, [7],

$$y_i^m = y(ih_m) + \sum_{q=1}^p e_q(ih_m)h^q + \mathcal{O}(H^{p+1}). \quad (14)$$

can successively be eliminated by an extrapolation technique until a method of a required order is obtained. The resulting method is known to be a Runge-Kutta method.

We exemplify the approach by aiming for third-order accurate Nordsieck values and restricting ourselves to autonomous differential equations for simplicity. The same process can be employed to obtain higher order accuracy.

Let  $h = \frac{H}{m}$ , and integrate the autonomous form of the differential equation (1) on the interval  $[y_0, y_0 + H]$  with Euler's method, using  $m$  steps of size  $\frac{H}{m}$  for  $m = 3, 2, 1$ .

For  $m = 3$

$$\begin{aligned} y_1^3 &= y_0 + hf(y_0) = y_0 + k_1, & k_1 &= hf(y_0), \\ y_2^3 &= y_1^3 + hf(y_1^3) = y_0 + k_1 + k_2, & k_2 &= hf(y_1^3), \\ y_3^3 &= y_2^3 + hf(y_2^3) = y_0 + k_1 + k_2 + k_3, & k_3 &= hf(y_2^3). \end{aligned} \quad (15)$$

For  $m = 2$

$$\begin{aligned} y_1^2 &= y_0 + \frac{3}{2}hf(y_0) = y_0 + \frac{3}{2}k_1, \\ y_2^2 &= y_1^2 + \frac{3}{2}hf(y_1^2) = y_0 + \frac{3}{2}k_1 + \frac{3}{2}k_4, & k_4 &= hf(y_1^2). \end{aligned} \quad (16)$$

For  $m = 1$

$$y_1^1 = y_0 + 3hf(y_0) = y_0 + 3k_1. \quad (17)$$

with  $h = \frac{H}{3}$ . We use approximation formulas for higher derivatives

$$\begin{aligned} h^k y^{(k)}\left(\frac{H}{2}\right) &= \sum_{i=0}^m d_{ik} y(ih_m) + \sum_{s=k+1}^p c_{sk} h_m^s y^s\left(\frac{H}{2}\right) \\ &\quad + \mathcal{O}(H^{p+1}), \quad m \geq k \end{aligned}$$

and (14) to obtain, after some algebraic manipulations,

$$\begin{aligned} D_3^3 &= y_3^3 - 3y_2^3 + 3y_1^3 - y_0 = h^3 y^{(3)}, \\ D_2^3 &= y_3^3 - y_2^3 - y_1^3 + y_0 = 2h^2 y^{(2)} + 2h^3 e_1^{(2)}, \\ D_2^2 &= y_2^2 - 2y_1^2 + y_0 = \left(\frac{3h}{2}\right)^2 y^{(2)} + \left(\frac{3h}{2}\right)^3 e_1^{(2)}, \\ D_1^3 &= y_3^3 - y_1^3 = hy^{(1)} + h^2 e_1^{(1)} + h^3 e_2^{(1)} + \frac{h^3}{24} y^{(3)}, \\ D_1^2 &= y_2^2 - y_0 = 3hy^{(1)} + \frac{9}{2}h^2 e_1^{(1)} + \frac{27}{4}h^3 e_2^{(1)} + \frac{9}{8}h^3 y^{(3)}, \\ D_1^1 &= y_1^1 - y_0 = 3hy^{(1)} + 9h^2 e_1^{(1)} + 27h^3 e_2^{(1)} + \frac{27}{24}h^3 y^{(3)}. \end{aligned} \quad (18)$$

All derivatives are evaluated at  $\frac{H}{2}$  and  $\mathcal{O}(h^4)$  terms are dropped. Estimates of the derivatives can be derived at any point within a constant multiple of the interval  $H$  with the same accuracy. We solve the system (18) to remove the error terms for  $h^k y^{(k)}\left(\frac{H}{2}\right)$ , for  $k = 1, \dots, m$ , to get

$$\begin{aligned} h^3 y^{(3)}\left(\frac{H}{2}\right) &= D_3^3 + \mathcal{O}(h^4), \\ h^2 y^{(2)}\left(\frac{H}{2}\right) &= \frac{3}{2}D_2^3 - \frac{8}{9}D_2^2 + \mathcal{O}(h^4), \\ hy^{(1)}\left(\frac{H}{2}\right) &= \frac{9}{2}D_1^3 - \frac{4}{3}D_1^2 + \frac{1}{6}D_1^1 + \frac{9}{8}D_3^3 + \mathcal{O}(h^4). \end{aligned} \quad (19)$$

The  $D_k^m$  can be expressed as combinations of stage values  $k_i$ . All  $\mathcal{O}(h^4)$  terms are neglected.

$$\begin{aligned} D_3^3 &= k_1 - 2k_2 + k_3, \\ D_2^3 &= -k_1 + k_3, \\ D_2^2 &= -\frac{3}{2}k_1 + \frac{3}{2}k_4, \\ D_1^3 &= k_2, \\ D_1^2 &= \frac{3}{2}k_1 + \frac{3}{2}k_4, \\ D_1^1 &= 3k_1. \end{aligned} \quad (20)$$

From (19) and (20),

$$\begin{aligned} h^3 y^{(3)}\left(\frac{H}{2}\right) &= k_1 - 2k_2 + k_3 + \mathcal{O}(h^4), \\ h^2 y^{(2)}\left(\frac{H}{2}\right) &= -\frac{1}{6}k_1 + \frac{3}{2}k_3 - \frac{4}{3}k_4 + \mathcal{O}(h^4), \\ hy^{(1)}\left(\frac{H}{2}\right) &= -\frac{3}{8}k_1 + \frac{9}{4}k_2 + \frac{9}{8}k_3 - 2k_4 + \mathcal{O}(h^4), \\ y\left(\frac{H}{2}\right) &= y_0 + \frac{3}{16}k_1 + \frac{18}{8}k_2 + \frac{9}{16}k_3 - \frac{12}{8}k_4 + \mathcal{O}(h^4). \end{aligned} \quad (21)$$

The first element of the Nordsieck vector,  $y\left(\frac{H}{2}\right)$ , is computed by Taylor expansion.

It follows that

$$\Gamma_{\frac{H}{2}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{16} & -\frac{3}{8} & -\frac{1}{6} & 1 \\ \frac{18}{8} & \frac{9}{4} & 0 & -2 \\ \frac{9}{16} & \frac{9}{8} & \frac{3}{2} & 1 \\ -\frac{12}{8} & -2 & -\frac{4}{3} & 0 \end{pmatrix}. \tag{22}$$

If the derivatives are instead computed at the origin, the matrix above becomes

$$\Gamma_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -\frac{5}{3} & 1 \\ 0 & 0 & 3 & -2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{4}{3} & 0 \end{pmatrix}. \tag{23}$$

The matrix  $A$  of coefficients  $\alpha_{i,j}$  in equation (2) is obtained from Equations (15) and (16)

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \frac{3}{2} & 0 & 0 & 0 \end{pmatrix}. \tag{24}$$

For a fourth-order method we need at least six function evaluations, [5], and the relevant matrices  $\Gamma_0$  and  $A$  are

$$\Gamma_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{5}{6} & \frac{4}{9} & -\frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{4}{9} & \frac{1}{9} \\ 0 & 0 & \frac{7}{3} & -\frac{19}{9} & \frac{7}{9} \\ 0 & 0 & -3 & \frac{10}{3} & -\frac{4}{3} \\ 0 & 0 & 1 & -\frac{11}{9} & \frac{5}{9} \end{pmatrix},$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ \frac{3}{4} & 0 & \frac{9}{4} & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 2 & 0 & 0 \\ \frac{1}{12} & 2 & \frac{1}{4} & \frac{2}{3} & 2 & 0 \end{pmatrix}.$$

The cost of this process in terms of function evaluations is  $1 + \frac{p(p-1)}{2}$ , since the interval  $H$  is integrated by Euler's method  $m$  times with step size  $\frac{H}{m}$  for

$m = p, p - 1, \dots, 1$ . For the first value of  $m$  we have  $p$  function evaluations because the initial value of  $y'$  has to be evaluated once, so for the next value of  $m$  we have  $p - 2$  function evaluations, and so on.

We constructed a Nordsieck vector with third-order accuracy. To do this we used four stages  $k_1, k_2, k_3, k_4$  as in (15) and (16) with lower order and an extrapolation technique. It can be shown that there exists no method of the same order with less stages and thus less function evaluations.

### 5 Order tests

To verify that the starter indeed achieves the expected order we consider the harmonic oscillator

$$y'' = -4y, \quad y_0 = 1, \quad y'_0 = 0.$$

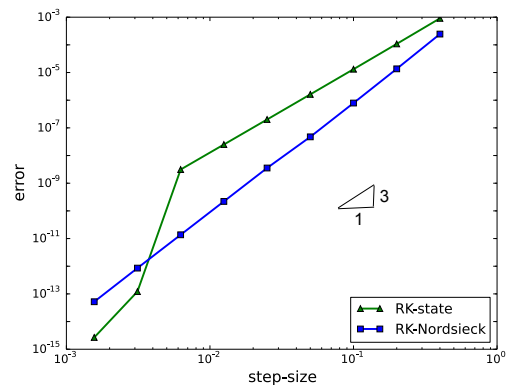


Figure 2: Both Runge–Kutta starters achieved third-order accuracy when solving the harmonic oscillator problem with the 3-step Adams–Moulton method

### 6 The bouncing ball test example

In this section we demonstrate the method on the example of a bouncing ball with linear damping  $d = 0.1$ :

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= -dy_1 + 9.81 \end{aligned}$$

The bounces are modeled using a coefficient of restitution was chosen to be  $c = 0.88$  to give the system sufficiently many impacts to be able to make a statement about the effect of restarting, see Fig. 3. The model includes two events, one to trigger bouncing and a second one which triggers the upper turning point. At the upper turning point the differential equation and its states remain unaltered, and only



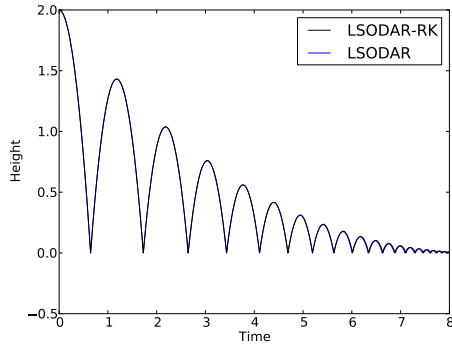


Figure 3: A simulation of a bouncing ball (damping:  $d = 0.1$ , coefficient of restitution  $c = 0.88$ ).

the switch to control the bouncing event becomes activated. At the bouncing event the velocity  $\dot{y}(t^-)$  is altered to  $\dot{y}(t^+) = -c\dot{y}(t^-)$ .

In Fig. 4 the step size and order history of both restarting techniques is compared. The classical starting procedure clearly shows a drop in order and step size. The method recovers quite quickly from the reduced step size as LSODAR allows exceptionally big step size changes during the initialization phase.

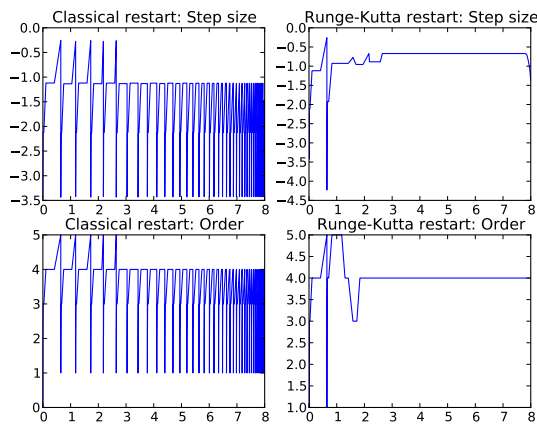


Figure 4: Comparison of the step size and order history for the two restarting approaches. A logarithmic scale is used for the step size plot.

The run statistics, cf. Tab. 1 show the effect of the Runge–Kutta starter in Assimulo. The gain in the number of function evaluations for this example is about 58%.

## 7 Conclusions

The aim of this paper is to study the effect of Runge–Kutta restarting techniques on the performance of the

	Classic starter	Runge–Kutta starter
# steps	455	129
# function evals	1027	428
# event function evals	919	538
# events	38	37

Table 1: Run time statistics for the bouncing ball example with absolute and relative tolerance set to  $10^{-8}$ .

simulation of hybrid systems. Tests were made on a system with relatively small numbers of discontinuities. The tests give a clear indication that investigating a more sophisticated restarting procedure like the fourth-order Runge–Kutta starter presented here has a potential impact on the overall performance of a simulator.

The flexibility in selecting the order of the restarter as well as doing error control of the restarter is the topic of future research.

## References

- [1] Johan Åkesson, Magnus Gäfvert, and Hubertus Tummescheit. JModelica—an open source platform for optimization of modelica models. In *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, February 2009. TU Wien.
- [2] Christian Andersson. Assimulo: a new Python based class for simulation of complex hybrid DAEs and its integration in JModelica.org. Master’s thesis, Lund University, 2011.
- [3] Christian Andersson, Johan Åkesson, Claus Führer, and Magnus Gäfvert. Import and export of functional mock-up units in JModelica.org. In *8th International Modelica Conference 2011*, Dresden, Germany, March 2011.
- [4] Torsten Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmquist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, et al. The functional mockup interface for tool independent exchange of simulation models. In *Modelica’2011 Conference, March*, pages 20–22, 2011.
- [5] C. W. Gear. Runge–Kutta starters for multistep methods. *ACM Trans. Math. Softw.*, 6(3):263–279, September 1980.

- [6] Anthony Ralston. Runge–Kutta methods with minimum error bounds. *Mathematics of computation*, 16(80):431–437, 1962.
- [7] Hans J. Stetter. Asymptotic expansions for the error of discretization algorithms for non-linear functional equations. *Numerische Mathematik*, 7(1):18–31, 1965.
- [8] Reinhold von Schwerin and Hans Georg Bock. A Runge–Kutta starter for a multistep method for differential-algebraic systems with discontinuous effects. *Applied numerical mathematics*, 18(1):337–350, 1995.

# Discontinuities handled with events in Assimulo, a practical approach

Emil Fredriksson\* Christian Andersson\*,\*\* Johan Åkesson\*,\*\*\*

\*Modelon AB, Sweden

Lund University, Sweden

\*\*Department of Numerical Analysis

\*\*\*Department of Automatic Control

## Abstract

Often integrating ordinary differential equations or differential algebraic equations (DAE) do not constitute the problem alone. A common complement is finding the root of an algebraic function (an event function) that depends on the states of the problem. This formulation of a model enables the possibility of including discontinuities, an important part of the Functional Mock-up Interface standard which allows hybrid models of differential algebraic equations. The problem of root-finding during integration is however difficult. Both in a theoretical aspect and as a software problem.

An implementation of software for root-finding is done in Assimulo, a Python/Cython wrapper for integrators. The implementation takes the Functional Mock-up Interface standard into consideration. The implementation is made usable for a wide variety of integration algorithms and is also verified and benchmarked with advanced industrial models, showing good indications of being robust and scaling well for large systems.

*Keywords: FMI; JModelica.org; Assimulo; events; discontinuities; Illinois algorithm; safeguard*

## 1 Introduction

Models based on differential equations may contain discontinuities. One simple example is the bouncing ball. Gravity acting on the ball is modelled with a differential equation while the bouncing on the floor will

result in discontinuities in the velocity. A result from the velocity changing sign by impact. A reasonable way to model this would be to restart the integration of the differential equation with new initial values as the ball hits the floor. In this way, the discontinuity is modelled with what is called an event and the handling of that event (event handling).

Models with discontinuities are not only interesting theoretically but are also widely used in industry, something the Functional Mock-up Interface (FMI) standard <sup>1</sup> contributes to by making distribution and use of these models convenient. The explanation for why it is used by the industry can be found in [3], where the elements that give rise to discontinuities in models are listed. Some of them are:

- Friction
- Impact phenomena
- The degrees of freedom vary in time
- Time dependent input functions

Most advanced models in industry consist of many separate but interacting parts and, therefore, have at least one or some of the listed properties. With today's modelling tools and computational power, more and more advanced models become realistic to simulate. Which means new and increased demands on the integrators to support the solving of models with discontinuities robustly and with good scaling of the performance regarding the size of the models.

There are many difficulties with having discontinuities in differential equations. Missing a discontinuity or acting on the wrong discontinuity can be disastrous,

<sup>1</sup>See <https://www.fmi-standard.org/>.

The authors gratefully acknowledge the support from the Lund Center for Control of Complex Engineering Systems (LCCC).

leading to integrating the wrong equations or missing impacts. Furthermore the integration methods make assumptions on the smoothness of the solution and the incorrect handling of the discontinuities will most certainly violate these assumptions. The result will be an incorrect error estimate, leading to a significant decrease in the integration performance[6], or even leading to an incorrect solution.

To construct a state of the art event detection algorithm, undertaking these considerations of the need for correct event handling and performance, the demands will be that it should be robust and scale well, handling large systems originating from industry. It should also be clear what data is expected from the user, and in return the algorithm should guarantee correct event detection and event handling.

The contributions of this article is an implementation of an event algorithm with robust event handling and good performance. Using a safeguard and applying the domain formulation (used for event localization in the FMI specification, explained in Section 2.4) as opposed to the zero-crossing paradigm that uses a sign change to detect events. This algorithm converge and gains a robust performance and has an advantage for a special set of problems, this will be seen for the clutch example later. Given the additions to the event algorithm, a benchmark using advanced industry relevant models following the FMI standard was made to ensure that the performance is not compromised. In Assimulo<sup>2</sup> the algorithm can be utilized as a module that can be mounted onto solvers as needed. This result in an extension of Sundials and increases the number of solvers that can handle discontinuities and therefore the number of solvers that support the FMI standard.

Section 2 starts by giving background and motivation and moves on to highlighting the principles of event detection and event localization and the difficulties associated with it. In Section 3, the ideas for the event location algorithm are laid out. Leading up to the presentation of the algorithm in Section 4, where details of the implementation are discussed. Section 5 demonstrates and verifies the implementation on a number of test examples together with testing the performance. A summary and critical examination of the algorithm and the results are given in Section 6.

<sup>2</sup>See <http://www.jmodelica.org/assimulo>.

## 2 Background

Assimulo is a Cython/Python wrapper around various Ordinary Differential Equations (ODE) and Differential Algebraic Equations (DAE) solvers. An important aspect of Assimulo is to make it easy to access both state of the art solvers and more experimental solvers for both industry and the academic world. Assimulo is also the back-end simulation engine for JModelica.org.

On the other hand, for enabling the exchange of models in industry and the academic world there is the FMI standard, which is a standardized way of formulating models of ODEs. PyFMI can be used to wrap models that are instances of the FMI standard (FMUs) making them easy to simulate with Assimulo.

A powerful use of Assimulo and the FMI would be to give industry a larger variety of solvers from the academic world, while the academic world is given access to a large number of relevant models from industry, offering remedies for two weaknesses often present in the world of numerics.

The FMI allows advanced hybrid dynamic models by also allowing event functions. This standard therefore demands that the solver can handle discontinuities in the form of events. Currently, the only solvers in Assimulo that can do so are those of Sundials [12] and LSODAR and many solvers do not have the possibility of handling discontinuities on their own, leading to the need of a module in Assimulo that can handle the discontinuities for all solvers.

### 2.1 A motivating example

A motivational example for when the zero-crossing approach fail to detect the event correctly is:

```
model motivating_example
Real y;
Real x(start = 1.0);

equation
  y = noEvent(if 1-time > 0 then (1-time)^5
              else 0);
  der(x) = if y <= 0 then -x else x;
end motivating_example;
```

The model has a variable  $y$  that smoothly goes to zero at  $t = 1$  which should result in an event there. With the usual approach for detecting events, using an FMU from JModelica.org, the event is found significantly later than it occurred because the event is not

localized. This is especially a problem when the integrator take large steps. It is clear that the robustness of the event detection is questionable in for this case.

With an FMU from Dymola the event is not detected at all.

## 2.2 The theory of event location

Integration methods are dependent on that the problem has a continuous solution with continuous derivatives to a certain order [3]. Moreover, mathematically, continuity is needed to guarantee a unique solution. For example for the problem of an explicit ODE

$$\dot{y}(t) = f(t, y(t)), \quad y(t_0) = y_0,$$

$f$  should be continuous in  $t$  and Lipschitz continuous in  $y$  to guarantee a unique solution by the Picard-Lindelöf theorem. This is a big concern with multi-step methods, which will incorrectly use information before and after discontinuities if the problem is integrated straight forwardly without explicit event handling.

To avoid integrating over a discontinuity, the event formulation can be used. One way of looking at a problem formulated with events is to imagine that it has two states, each state representing a different continuous right hand side. The discontinuity then becomes switching between these two states (an event). More formally, the point in time of switching is decided by the sign change of an event function,  $g(t, y)$ , and at this time the integration is re-initialized with updated continuous variables and discrete variables, where the different states are represented by discrete variables that only change at events. For example, the discontinuous problem:

$$\dot{y} = f(t, y) = |y|,$$

is rewritten with the event formulation as:

$$\dot{y} = f(t, y, s) = \begin{cases} y & \text{if } s = s_1 \\ -y & \text{if } s = s_2, \end{cases}$$

$$g(t, y, s) = y,$$

where  $g$  and  $f$  are continuous for a fixed value of the discrete state  $s$  (that represent a state). The downside being that this add a root-finding problem for the event function  $g$  on the interval of the latest time steps,  $t_n$  and  $t_{n+1}$ , in case an event is detected.

What is described in this section is usually called discontinuity handling. It can intuitively be divided into three steps:

- Event detection
- Event localization (in  $[t_n, t_{n+1}]$ )
- Event handling

The detection is often done by checking the sign of  $g$  after every time step. Locating the event is done with a root-finding algorithm and the event handling is mainly a modelling question that is done by the user.

The integrating of ODEs with discontinuities has received a lot of attention over the years. Many of the differences between the approaches is how  $g$  will be represented on  $[t_n, t_{n+1}]$  and how the time of the event is localized.

Most of the methods for localizing the event require the ability to evaluate  $g(t, y)$  on  $[t_n, t_{n+1}]$ . In doing so effectively, a continuous extension of  $y(t)$  on  $[t_n, t_{n+1}]$  is desired. Not using a continuous representation when solving problems with discontinuities result in larger global error and more evaluations of  $f$ , see [4]. Also, the dependence between the global error and the tolerance was smoother for problems with discontinuities when using an interpolation polynomial for root-finding.

Further theoretical results strengthening the use of interpolation polynomials for problems with discontinuities are that if the interpolation polynomial is of the same order as the integration method the entire method has this order [16].

Besides the representation of  $y$  with an interpolation polynomial, there is the idea that additional states could be introduced through new state equations of the form  $\dot{y}_{n_y+1:n_y+n_g} = \dot{g}$ , where  $n_y$  is the dimension of  $y$  and  $n_g$  is the dimension of  $g$  [2] [17]. This will force the integrator to take steps such that the dynamics of  $g$  is captured, if this is not the case it is more likely that an event will be missed due to two changes in sign of  $g$  are canceled out or that not the first event of many on  $[t_n, t_{n+1}]$  is found.

### 2.2.1 The root-finding problem

The root-finding problem for localizing the event with the event function,  $g$ , have the properties that  $g \in C^0$  (on a bounded interval  $[a, b]$ ) and  $g(a)g(b) < 0$ . Through the intermediate value theorem, the existence of a zero in the interval  $[a, b]$  is guaranteed. Due to the nature of solving the problem numerically, the zero can often not be found exactly. Therefore the problem is said to be solved if an interval  $[a^*, b^*]$  is found, such that:

$$g(a^*)g(b^*) < 0 \text{ and } |a^* - b^*| < \delta.$$

This means that the zero of  $g$  is contained in a small interval of length  $\delta$ . Also, note how the condition  $g(a)g(b) < 0$  functions as an enclosing property; this will from here on be known as a regula falsi. The goal is to find an algorithm that always converge and doing so as quickly as possible for a wide range of functions. The goal can be considered delicate because of the large spectrum of functions that are allowed.

## 2.3 Integrators in Assimulo

Presented here are some of the ODE integrators wrapped by Assimulo that are suited to be used together with an event localization algorithm and therefore use to simulate FMUs. Their types, orders and interpolation are listed.

**Explicit and Implicit Euler:** Fixed step-size methods of order 1 with linear interpolation implemented.

**RungeKutta34:** Adaptive Runge-Kutta of order 4(3) with a Third-order Hermitian polynomial for interpolation.

**Radau5ODE:** Runge-Kutta method based on Radau IIA of order 5, with interpolation from its collocation solution. The interpolation polynomial is of order 3 [9] [10].

**Dopri5:** Runge-Kutta method, is of order 5(4) with an interpolation of order 4 [9] [10].

**RodasODE:** A Rosenbrock method of order 4(3). The order of the interpolation is not stated explicitly, but it is said to fulfil conditions such that the continuous solution is of the same order as the discrete points. Uses variable step-size [9] [10].

**CVode:** Uses BDF methods for stiff problems and Adams-Moulton methods for non-stiff problems. For both cases, the solver is of variable-order and has variable step-size. Contains an internal event localization algorithm [12].

## 2.4 FMI semantic and domain formulation

The condition that an event occurs when  $g$  changes sign (a change between  $g < 0$  and  $g > 0$ , zero-crossing formulation) means that the zero needs to be treated as an exception. An option is that one should instead look for an alternation between the domains  $g < 0$  and  $g \geq 0$  [14]. This leads to a formulation similar to the event formulation in the FMI standard, where the enclosing and detection properties (regula falsi) change

from  $g(a)g(b) < 0$  to  $(g(a) > 0) \oplus (g(b) > 0)$ <sup>3</sup>.

## 3 Event algorithm

### 3.1 Domain or zero-crossing formulation

The arguments for using the domain formulation, in addition to being consistent with the FMI standard, are that the zero is no longer a special case. [11].

The FMI formulation also has a major advantage when modeling systems that can take an unknown input. Let us suppose that  $g(t, y) = u(t)$ , where  $u$  is a signal that can become and stay at zero. A practical example would be if  $u$  is the power to a system. Imagine now that the system as a safety measure has a magnet-locking system or clutches that locks when the power disappears. An event is then expected when  $u > 0$  goes to  $u \leq 0$  or vice versa. This is an intuitive way to state the model and would, with the zero-crossing formulation, force the user to modify  $g$  or the inequality with a small  $\varepsilon$  to ensure zero-crossing. Moreover, the choice of  $\varepsilon$  is often not an easy task in this case because of scaling.

### 3.2 Event detection

One of the usual ways to detect an event is to check the regula falsi for  $g(t_n, y_n), g(t_{n+1}, y_{n+1})$  after having integrated from  $t_n$  to  $t_{n+1}$ . Other ways are, of course, possible - such as also checking  $g$  in the middle of  $[t_n, t_{n+1}]$  - but these are considerably slower. This is the case for many of the more sophisticated methods for detecting events and they furthermore demand access to the partial derivatives of  $g$ . This is also the case for the methods of adding extra states. Demanding the user to supply these or compute them numerically, giving the user a solver that scales badly (computing the derivatives numerically would also result in extra evaluations of  $g$ ) is not an option. It does not align with our demands of speed, it would also exclude models of the FMI standard.

Going with the simpler method of checking for a regula falsi, there is the possibility of having two events in  $[t_n, t_{n+1}]$  for a component of  $g$ . This is a problem, as pointed out in Section 2.2. The practical solution used here lies in letting the user supply a maximum stepping length,  $h_{\max}$ , such that all events are separated by at least  $h_{\max}$  in time.

<sup>3</sup>The logical symbol  $\oplus$  is XOR and in code the condition would be  $(g(a) > 0) \neq (g(b) > 0)$

### 3.3 Locating the event

A root-finding algorithm that is usually used in this context is the Illinois algorithm [5] [13] [12]<sup>4</sup>. Naturally, it gives fast convergence for most functions when doing event localization. There are of course other root-finding algorithms that would perform well for event localization, Illinois is however well tested. Furthermore, if  $g$  is multidimensional and the first root in time should be found for any of the components of  $g$  and this vectorization is best done for the Illinois algorithm.

The Illinois algorithm uses a linear interpolation that weights the function values to ensure convergence. The weight is applied so that the algorithm will not keep any bracket constant indefinitely. This gives major advantages when it comes to convergence compared to the false position algorithm. This is especially evident for all convex functions, a comparison can be seen in Figure 1.

An improvement can be done for when the function is zero for most of the interval as Illinois algorithm behaves badly in this case. This comes from the Illinois algorithm's inability to use its weights properly - zero times two is still zero. The intuitive remedy for this would be to use a bisection step if either  $g(t_n)$  or  $g(t_{n+1})$  are equal to zero. With this modification, the Illinois algorithm behaves better for this case.

#### 3.3.1 Safeguard

To ensure that the algorithm converges a safeguard method is used [7]. It consist of three points that, if followed, guarantee that the root-finding algorithm always converge. These are:

- The new bracket is inside the current interval
- The new bracket is closest to the best bracket<sup>5</sup>
- The new bracket is not too close to an existing bracket

Point one is incorporated in the Illinois algorithm. Point two is not always valid during the iterations, due to the weights. Point two is however only meant to ensure the speed of convergence, something that the weights do very well.

<sup>4</sup>The origin of the method is unknown - some believe it to come from the staff at the computer centre at the University of Illinois computer department.

<sup>5</sup>The best bracket is the one with the lowest function value, as it is expected to lie closer to the root.

Point three is often done by choosing a small number  $\delta$  and check if the new bracket is closer than  $\delta$  to a previous bracket, if it is the new bracket is moved by  $\delta$  towards the middle. It should be noted that Illinois algorithm dose not originally account for this. There is, however, no technical problem in implementing this (Sundials has this modification in its implementation).

## 4 Implementation

### 4.1 User interaction

The form of the event function  $g$  defined by the user should be a function returning an array of all the components of  $g$ . This is also how the event function is evaluated in the FMI standard, but results in that all components are evaluated, even if only one component has a change in sign when iterating with the Illinois algorithm. However, this gives the possibility of detecting roots that otherwise would have been missed due to being close to another root.

Also, the reporting of the found roots should be done with a separate array. The user cannot know for which component a root was detected in the case where the exact zero is not found due to scaling.

The tolerance for enclosing the event cannot be set and has a default value such that the error caused by it is small compared to that caused by the interpolation polynomial.

### 4.2 Algorithm details

The tolerance for which the domain change is locked into is somewhat weaker than machine precision. Specifically, the tolerance is set to:

$$TOL = \max(|t_n|, |t_{n+1}|) \cdot 10^{-13},$$

being roughly a factor 100 times larger than machine precision, it still does not introduce an error that is noticeable compared to that of the interpolation polynomial.

The small step  $\delta$  in the safeguard (see Section 3.1.1) is taken from Sundials as:

$$\delta = \frac{|a - b|}{2 \cdot \min(5, |a - b|/TOL)},$$

note that  $\delta$  is a minimum of a tenth of the current length of the interval and a maximum of half the interval, going from a tenth to a half as the interval goes toward TOL.

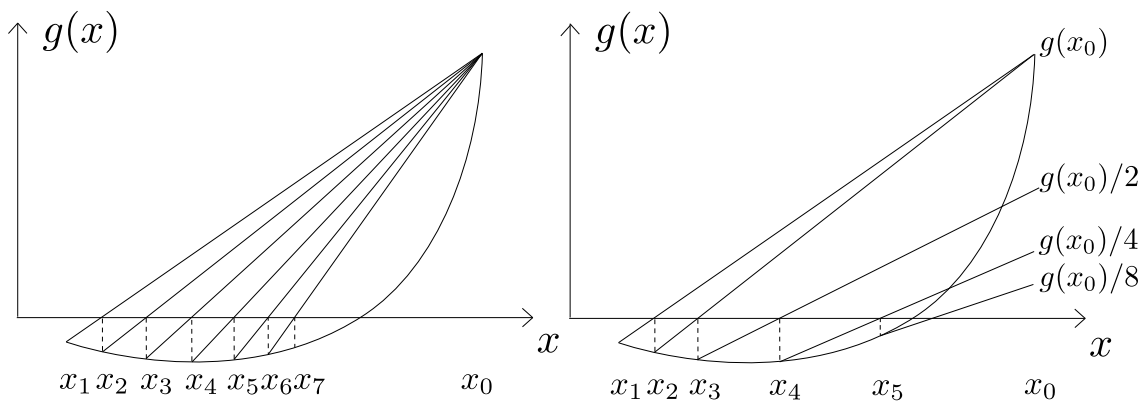


Figure 1: Comparison between the Illinois algorithm and the false position method, showing how the failing mode is escaped and fast convergence is achieved.

The domain formulation and the bisection step for the special case mentioned earlier are used by the algorithm. The algorithm is implemented in Assimulo using Python. It is called before the complete step function is called just after a successful step is taken.

### 5 Application example

For evaluating the implementation, benchmark models are selected to assert the correctness and scalability of the event location algorithm. A Furuta pendulum is chosen for having many event functions, a clutch model with inputs is chosen for causing events when the input reach zero for a finite time and a racing car is chosen for being a large advanced model with events.

The models are FMUs and PyFMI is used for simulating these models in Assimulo. Benchmarks are made for each model and each ODE solver that the event location is implemented for. The quantities included in the benchmark are: steps taken, function evaluations, Jacobian evaluations, event-function evaluations, number of events and simulation time in seconds.

The options for the solvers are tuned so that the solution at the final step is found with a relative error of roughly  $10^{-6}$  compared to a reference solution, with  $10^{-6}$  here roughly meaning that the relative error is in the interval  $[10^{-6}, 2 \cdot 10^{-6}]$ , preferably as close to the left bracket as possible. Formally, this means that the solution should satisfy the condition:

$$10^{-6} \leq \frac{\|y(t_{\text{final}}) - y_{\text{ref}}(t_{\text{final}})\|_2}{\|y_{\text{ref}}(t_{\text{final}})\|_2} \leq 2 \cdot 10^{-6}.$$

This error is foremost tuned by changing the relative- and absolute-tolerance. In case the integrator takes too

large steps and events therefore are missed, a maximum step length is also used to tune the accuracy of the solution. For RungeKutta34, the optional initial step length is also used - RungeKutta34 would otherwise have a problem reducing the tolerance sufficiently, as it has no possibility to reject its first step. The other steps are affected by the tolerance through the variable step-size (even though the step cannot be rejected).

For finding the reference solution, CVode with its internal event location is used. The options for relative and absolute tolerance is set to  $10^{-12}$ . At this tolerance, the relative error estimate does not change if the tolerance is increased or decreased by a factor 10, meaning that significantly more correct decimals are found for this solution compared to the solutions satisfying the condition on relative error.

An important reason for using CVode is that it is extensively tested and well established.

Following the numerical results there is a discussion of them. In the discussion, important differences between the solvers are pointed out, and extra emphasis is placed on the differences between CVode with internal event location and CVode with Assimulos event location, here after known as CVode(I) and CVode(A).

#### 5.1 Furuta pendulum

The Furuta pendulum (see Figure 2), generated by Dymola, is an extremely non-linear model and is often investigated in the field of control theory [18]. This model of the problem generates events from introduced friction, resulting in a problem with 32 event functions. When simulated for 5 seconds, 21 events occur for this problem, making it a good test model for the.



Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$2 \cdot 10^{-8}$	$3.1 \cdot 10^{-8}$	$1.81 \cdot 10^{-6}$	$1.42 \cdot 10^{-6}$	$1.8 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
Absolute tolerance	$2 \cdot 10^{-8}$	$3.1 \cdot 10^{-8}$	$1.81 \cdot 10^{-6}$	$1.42 \cdot 10^{-6}$	$1.8 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
Initial step length	-	-	-	-	-	$10^{-6}$

Table 1: The solver options to obtain the desired accuracy for the Furuta pendulum. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1107	1050	113	329	231	589
f evaluations	1468	1430	776	2109	1680	2945
J evaluations	34	30	-	329	156	-
g evaluations	1332	1266	372	615	504	787
Execution time	2.760	2.888	0.6183	3.165	0.9516	6.157
Relative error	$1.05 \cdot 10^{-6}$	$1.08 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$	$1.03 \cdot 10^{-6}$	$1.18 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$

Table 2: Run-time statistics for the solvers used on the Furuta pendulum. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

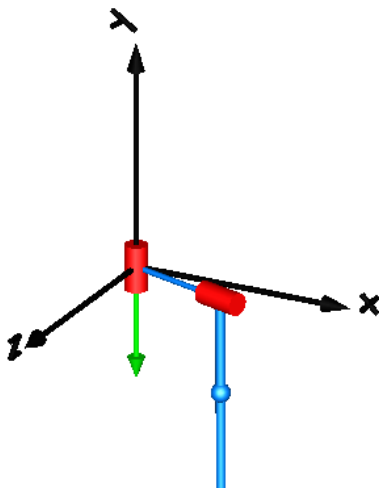


Figure 2: A Furuta pendulum. The red cylinders being joints with given frictions and the blue cylinders being bars with given weights.

The first thing to note is that CVode(A) achieves the same error as CVode(I) with higher tolerances, somewhat adding robustness to the solving process. Nonetheless, the execution time is longer despite using fewer function evaluations.

Dopri5 is the solver that performs best.

## 5.2 Clutches with input

This system is of practical industrial interest, where an input signal causes events when reaching zero for

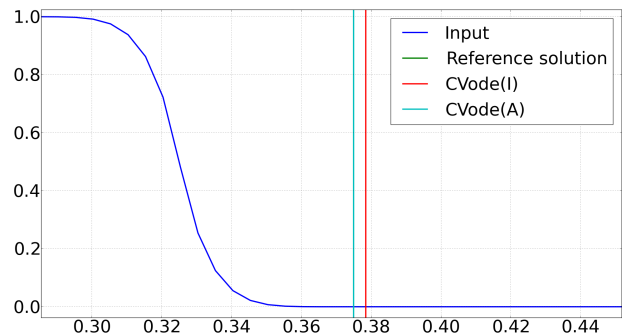


Figure 3: Plot of the input and where the first event is found for the different solvers. The event found by the reference solution and CVode(A) can not be distinguished in the figure.

a finite time, very much as described in Section 3.1. The number of events that occur for the 5 seconds that the system is simulated varies, as some of them are handled internally by the FMI's event iteration. However, all the events are detected and handled in one way or another, as indicated by the small relative errors the problem is solved for. The FMU is generated by JModelica.org.

The events caused by the input signals are located with close to machine precision for CVode(A) and only with the current step length when detecting  $g(t_{n+1}) = 0$  for CVode(I). Despite this, the same choices for tolerances give the same relative error. This is however still considered an

Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$6 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$4.39 \cdot 10^{-5}$	$5.033 \cdot 10^{-9}$
Absolute tolerance	$6 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$4.39 \cdot 10^{-5}$	$5.033 \cdot 10^{-9}$
Max step length	-	-	-	0.1	0.1	-
Initial step length	-	-	-	-	-	$10^{-10}$

Table 3: The solver options to obtain the desired accuracy for the Clutch system with the input signal. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1594	1594	656	305	233	5312
f evaluations	2568	2552	4498	2370	1275	26545
J evaluations	71	71	-	305	93	-
g evaluations	2480	2733	1876	1739	1746	6223
Execution time	8.476	8.425	17.56	7.709	6.756	55.42
Relative error	$1.11 \cdot 10^{-6}$	$1.07 \cdot 10^{-6}$	$1.16 \cdot 10^{-6}$	$1.08 \cdot 10^{-6}$	$1.24 \cdot 10^{-6}$	$1.09 \cdot 10^{-6}$

Table 4: Run-time statistics for the solvers used on the Clutch system with the input signal. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

important result, finding the events with good accuracy is desired and is expected to pay off for other models. The first event from the input signal at 0.0375 is found to be 0.375000000000015, 0.378400333934374, 0.375000000000025 for the reference solution, the CVode(I) and the CVode(A) respectively, a significant improvement as can be seen in Figure 3. Radau5 performs best for this model.

### 5.3 Racing car

A large and advanced model from industry is the racing car. Here, the model not only contains the racing car but also a virtual driver. It consists of a regulator that tries to drive the car on an eight-shaped course. One reason for simulating this might, for example, be to investigate the dynamic response of the car.

Consisting of 47 states, 44 event functions and simulated for 30 seconds, this is the largest model used for testing. During simulation, the model caused 11 events. The model is developed and generated from Dymola originating from the Vehicle Dynamics Library, see Figure 4.

Here, CVode(A) performs better than CVode(I), in the sense that it needs less tolerance for achieving the same error. This is a good sign of robustness and scalability.

For the racing car model, evaluations of f are more

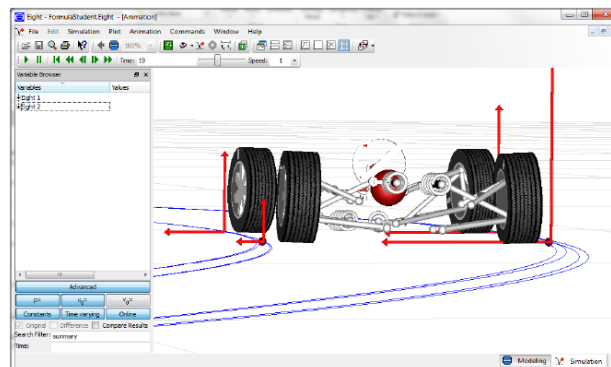


Figure 4: A picture of the car in Dymola.

costly compared to those in the Furuta model. This makes CVode the most effective integrator. Also note that Radau5 need the setting  $h_{\max} = 0.1$  for its maximum step length to capture all the events.

### 5.4 Summary

The event localization of Assimulo, using the domain formulation, finds the events caused by the inputs in the clutch model significantly better than Sundials, using the zero-crossing formulation.

If the event localization of Assimulo is compared by looking at the results for CVode(A) and CVode(I), it is clear that Assimulos event algorithm do not perform significantly worse than that of Sundials. For all

Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$5.5 \cdot 10^{-6}$	$10^{-5}$	$3 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	$3.5 \cdot 10^{-5}$	$7.3 \cdot 10^{-3}$
Absolute tolerance	$5.5 \cdot 10^{-6}$	$10^{-5}$	$3 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	$3.5 \cdot 10^{-5}$	$7.3 \cdot 10^{-3}$
Max step length	-	-	-	-	0.1	-
Initial step length	-	-	-	-	-	$10^{-9}$

Table 5: The solver options to obtain the desired accuracy for the racing car. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1248	1287	2163	333	358	2817
f evaluations	1720	1736	13152	2143	2330	14085
J evaluations	38	34	-	333	248	-
g evaluations	1485	1478	2409	539	638	3039
Execution time	14.45	14.59	23.76	21.44	17.78	27.49
Relative error	$1.31 \cdot 10^{-6}$	$1.54 \cdot 10^{-6}$	$1.39 \cdot 10^{-6}$	$1.57 \cdot 10^{-6}$	$1.17 \cdot 10^{-6}$	$1.31 \cdot 10^{-6}$

Table 6: Run-time statistics for the solvers used on the racing car model. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

models except the racing car CVode(A) uses more  $g$  evaluations than CVode(I). Sometimes, this results in better accuracy for the selected tolerances. This is the case for the Furuta pendulum and the racing car model. For the clutch model with an input signal, there is, on the other hand, nothing to be gained and all that is achieved is extra  $g$  evaluations.

The new solvers that now support event localization all performed reasonably. For this set of models CVode and also Dopri5 performs well, making it together with all the new solvers supporting event location a welcome addition to the toolbox of integrators supporting event location and therefore also FMUs.

For some of the models, it is noted that Runge-Kutta methods performs very well as indicated by the small number steps taken. This is made possible by the sparse occurrence of events and it is worth noting that none of the models here has events occurring with a high frequency. In [8], this was investigated using small balls bouncing around, receiving different frequencies for the events by changing the number of balls.

## 6 Summary and conclusions

It was found that an algorithm based on the Illinois algorithm works well. The improvements include applying the domain formulation, extra safeguarding and

the special case when  $g = 0$  resulting in a bisection step. It was implemented in Assimulo and was shown to locate certain types of events more accurately for an industry-relevant model with clutches than Sundials, without losing much in performance. Much of the performance lost is, of course, because Assimulo's event location is written in Python while Sundial's is written in C. Further improvements can be made reducing this difference by typing the variables using the possibilities of Cython in Assimulo, leading to the code executing more like C code, even though the differences in speed can probably never be fully remedied.

The implementation can be used together with many solvers as it is a separate subroutine, a module that is easily mounted onto solvers. Resulting in the possibility to choose among many solvers when simulating FMUs. Also, solvers with interpolation polynomials of lower order than the method itself still performed well, without any guarantees from the theory.

For the case of a problem containing a large number of event functions, the improvement is being able to choose Runge-Kutta methods, which are good at solving many of the models with sparse occurrence of events. In the case of the extreme opposite, many events occurred with high frequency, CVode with a  $h_{\max}$  matching this frequency would probably perform best.

More work for the future might include the possibil-

ity of supplying the partial derivatives of  $g$  or of calculating these numerically. Uses might be enhanced event location, as with [17] or [1]. Further work in investigating event location for implicit DAEs is also needed, for example implementing consistent event localization [15] and test the gain in accuracy and the loss in computational speed on relevant industry problems.

## References

- [1] CARVER, M. Efficient integration over discontinuities in ordinary differential equation simulations. *Mathematics and Computers in Simulation* 20 (1978), 190 – 196.
- [2] CARVER, M., AND MAC EWEN, S. Numerical analysis of a system described by implicitly-defined ordinary differential equations containing numerous discontinuities. *Applied Mathematical Modelling* 2 (1978), 280 – 286.
- [3] EICH-SOELLNER, E., AND FÜHRER, C. *Numerical methods in multibody dynamics*, corr. repr.; 2. corr. repr., 2008 ed. Teubner, [Lund], 2002.
- [4] ENRIGHT, W., JACKSON, K., NORSETT, S., AND THOMSEN, P. Interpolants for Runge-Kutta formulas. *ACM Transactions on Mathematical Software* 12, 3 (1986), 193–218.
- [5] FORD, J. *Improved Algorithms of Illinois-type for the Numerical Solution of Nonlinear Equations*. University of Essex, Department of Computer Science, 1995.
- [6] GEAR, C. W., AND OSTERBY, O. Solving ordinary differential equations with discontinuities. *ACM Transactions on Mathematical Software* 10, 1 (1984), 23.
- [7] GILL, P. E., MURRAY, W., AND WRIGHT, M. H. *Practical optimization*. Academic Press, London, 1981.
- [8] GRABNER, G., AND KECSKEMETHY, A. An integrated Runge-Kutta root finding method for reliable collision detection in multibody systems. *MULTIBODY SYSTEM DYNAMICS* 14, 3-4 (2005), 301 – 316.
- [9] HAIRER, E., NORSETT, S. P., AND WANNER, G. Solving ordinary differential equations i: Nonstiff problems (e. hairer, s. p. norsett, and g. wanner). *SIAM Review* 32, 3 (1990), 485.
- [10] HAIRER, E., AND WANNER, G. *Solving Ordinary Differential Equations II [Elektronisk resurs] : Stiff and Differential-Algebraic Problems / by Ernst Hairer, Gerhard Wanner*. Springer Series in Computational Mathematics: 14. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010., 2010.
- [11] HIEBERT, K., AND SHAMPINE, L. Implicitly defined output points for solutions of odes. *Sandia National Laboratory Report SAND80-0180* (1980).
- [12] HINDMARSH, A., BROWN, P., GRANT, K., LEE, S., SERBAN, R., SHUMAKER, D., AND WOODWARD, C. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software* 31, 3 (2005), 363–396.
- [13] MOLER, C. Are we there yet? zero crossing and event handling for differential equations. *EE Times Simulink 2 Special Edition* (1997), 16–17.
- [14] OLSSON, H. *Runge-Kutta solution of initial value problems : methods, algorithms and implementation / Hans Olsson*. Lund : Univ., 1998, 1998.
- [15] PARK, T., AND BARTON, P. I. State event location in differential-algebraic models. *ACM Transactions on Modeling and Computer Simulation* 6, 2 (1996), 137–165.
- [16] SHAMPINE, L., AND THOMPSON, S. Event location for ordinary differential equations. *Computers and Mathematics with Applications* 39, 5-6 (2000), 43–54.
- [17] SHAMPINE, L. F., GLADWELL, I., AND BRANKIN, R. W. Reliable solution of special event location problems for odes. *ACM Transactions on Mathematical Software* 17, 1 (1991), 11.
- [18] XU, Y., IWASE, M., AND FURUTA, K. Time optimal swing-up control of single pendulum. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* 123, 3 (2001), 518–527.

# Noise Generation for Continuous System Simulation

Andreas Klöckner  
andreas.kloeckner@dlr.de

Franciscus L. J. van der Linden  
franciscus.vanderlinden@dlr.de

Dirk Zimmer  
dirk.zimmer@dlr.de

German Aerospace Center (DLR), Institute of System Dynamics and Control,  
82234 Weßling, Germany

## Abstract

Adding random disturbances to Modelica models is necessary to represent stochastic fluctuations like sensor noise, air gusts and road irregularities. In this paper, we present a library to specify a pseudo random noise for continuous-time simulations. The random number generator, a probability density function and a frequency spectrum can be defined independently. A new random number generator is proposed to generate a continuous random signal, which is proven to be highly suitable for continuous models. The performance of the noise models is tested in two benchmarks using an academic as well as a realistic model both showing a remarkable increase in simulation speed.

*Keywords: Noise, Stochastic Models, Random Number Generator*

## 1 Motivation

When simulating real-world systems, the problem of introducing disturbances to the nominal system eventually becomes an issue. Especially, when dealing with controlled systems, important tasks are to check whether the controller is able to reject realistic disturbances, and to assess the performance of the system including noise. The problem is not limited to the field of control design, but is also of interest in e.g. specification of aircraft airworthiness requirements with respect to turbulence, estimation of the power outcome of wind energy farms or when interpreting contaminated sensor readings of experiments.

These kinds of distortions are typically taken into

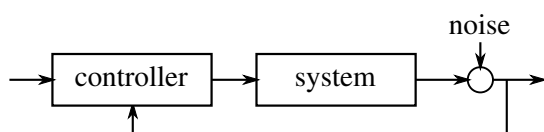


Figure 1: Noise is typically introduced additively into a controlled system.

account by additive injection of a noise signal into the system as shown in Fig. 1. The noise signal can have a strong impact on the system's performance and must thus be specified carefully. However, there are no convenient means of specifying noise properties in Modelica, such that typical approaches implement ad-hoc modifications of a simple random signal.

Additionally, injecting noise into a continuous system typically decreases the simulation speed drastically, because standard noise generators are sampled systems, which generate time events by definition. These time events lead in most cases to integrator restarts, imposing a big penalty on simulation performance.

In this work, we present ways to solve the two issues outlined above in an integrated library:

1. We describe a general procedure to specify a suitable noise signal by means of selecting a high-quality random number generator, a probability distribution and a power spectrum (see Sec. 3).
2. By providing a continuous noise signal formulation using the sample-free generators introduced in Sec. 3.1 and a smooth interpolation (Sec. 3.3), we provide means for continuous noise generation. Avoiding events and using a smoothly filtered signal speeds up the simulation as compared to standard methods (see Sec. 5).
3. The methods and processes are integrated in a library with convenient user interfaces (see Sec. 4). This enables a user to easily specify a desired noise signal and to use it in complex simulation models (see Sec. 6).

## 2 Theoretical Background of Noise

Noise is omnipresent in technical systems. However, it is not usually a physical process per se. The term noise is rather used to describe influences in a system, which are not covered by the model of the system itself.

These influences can e.g. include rough road conditions in vehicle dynamics, wind in aerodynamics or electrical radiation compromising sensor readings. In any case, a suitable model for such influences must be found in order to account for them in simulation, control and signal processing applications.

A common model for noise is *white noise* (see e.g. [1, p. 19]). It is described by a stream of random variables  $w(t)$  dependent on the time  $t$ . The main property of such white noise is that it has a flat power spectrum, i.e. that all frequencies contribute equally to the noise signal. In a statistical sense, this means that all instances of the random signal  $w(t_1)$  and  $w(t_2)$  are uncorrelated:

$$\text{Cov}(w(t_1), w(t_2)) \stackrel{!}{=} 0 \quad \forall t_1 \neq t_2. \quad (1)$$

The second property of white noise is that the probability distribution  $W$  of the signal's values is equal for all time instants:

$$W(t_1) \sim W(t_2) \quad \forall t_1, t_2. \quad (2)$$

However, the actual distribution  $W$  is not specified by the term white noise and must be specified additionally. Gaussian white noise e.g. refers to the case that the signal is normally distributed.

In actual applications, noise is very rarely white but can be specified with a given power spectrum. In aerodynamics e.g. the von Kármán spectrum is used to model turbulence and in signal processing the noise is usually low-pass filtered and sampled. Such signals are referred to as colored noise.

Especially for numerical simulations, this band-limitation of natural noise is very convenient, because signals with infinite frequency contributions cannot be simulated. Using the above observation, we can correctly reflect the influence of natural noise on a simulated system by specifying a distribution of the random process and a suitable filter. The sampling rate of the raw noise can then be chosen just high enough to generate all contributions to the desired spectrum.

Figure 2 illustrates the properties of natural noise from a mechanical test-rig entering a continuous system. The noise is sampled with 6 kHz and passed on as a continuous-time sample-and-hold signal. The power spectral density (PSD) is estimated by oversampling the signal at 50 kHz and using Welch's method as implemented in the Matlab signal processing toolbox.

The noise appears approximately normally distributed. We can see that the noise does not have the flat power spectrum specified for white noise. The power spectrum shows a strong influence of the initial sampling with 6 kHz. In the following sections, we will

discuss how to numerically generate a representative signal of such nature.

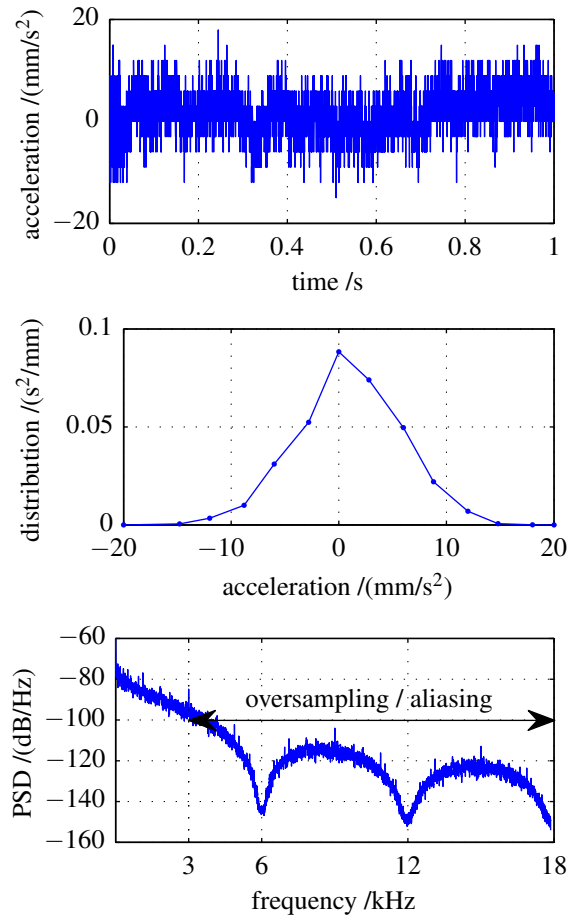


Figure 2: A noise sample from a static mechanical acceleration sensor is approximately normally distributed and has a characteristic power spectrum.

### 3 Noise Generation

In order to conveniently generate a realistic noise for use in continuous system simulations, a couple of criteria must be met:

- The noise should be *realistic* with respect to the specifications given in Section 2: The signal should be uncorrelated, match a specified distribution and also match the specified frequency content.
- A *clear and modular* approach should provide support in independently specifying these properties of the desired noise signal.
- It should be possible to evaluate the noise signal *continuously* without the need to incrementally increase the internal states as is the case with conventional noise generators.

To comply to the specifications of above, the process is split in three steps:

1. A *random number generator* (RNG) implements an algorithm to generate a sequence of uncorrelated, uniformly distributed random numbers  $U_i$ .
2. These are transformed according to the same *probability density function* (PDF) for each discrete time instance separately to yield random numbers  $X_i$  with a specified distribution.
3. The random numbers are filtered to match a *power spectral density* (PSD) specified in the frequency domain. This results in a continuous-time noise signal  $r(t)$  with the desired characteristics.

### 3.1 Uniform Random Number Generators

Truly random numbers are difficult to generate in numerical simulations. Fortunately, they are typically not desired to be truly random, because simulations should be repeatable and thus deterministic. Pseudo-random numbers are thus usually used, which are deterministic but appear random to the simulated system.

Pseudo-random numbers are usually generated computationally using recursive arithmetic generators. These generate random numbers  $Y_i$  recursively based in the last random number  $Y_{i-1}$ . The initial value  $Y_0$  is known as the seed of the recursive generator. This discrete state  $Y_i$  of the random number generator must be advanced incrementally, which limits the time steps taken by an integrator to be smaller than the generator's update period.

One of the simplest recursive arithmetic generators is the linear congruential generator (LCG). It uses the parameters  $a$ ,  $b$  and  $m$  to generate random integers  $Y_i$  in the interval  $[0, m-1]$  and uniformly distributed numbers  $U_i$  according to the following formulas:

$$Y_i = (a \cdot Y_{i-1} + b) \mod m, \quad (3)$$

$$U_i = Y_i/m. \quad (4)$$

Better implementations of recursive arithmetic generators are e.g. the algorithms of the WELL family (Well Equidistributed Long-period Linear) [2], which feature much larger repetition periods.

In order to avoid the performance limitation introduced by the discrete state of recursive generators, we propose to use non-recursive arithmetic generators for generating random numbers. These implement a pure function  $Y_i(t)$ , which is solely dependent on the simulation time  $t$ . This allows to evaluate random numbers deterministically in continuous time without using discrete states.

In this work, we introduce the new random number generator **DIRCS Immediate Random with Continuous Seed**. It relies on the quick recovery of LCGs from a poor (i.e. small, non-random) seed  $Y_0$ , a property called diffusion capacity. If a poor seed  $Y_0$  is chosen, an LCG will irrespectively generate high quality random numbers after a few iterations. This property allows to continuously seed an LCG with a very simple function of the time  $t$ , apply a few iterations and treat the resulting number as random, i.e.

$$Y_i(t) = LCG(\dots(LCG(\text{seed}(t)))) \quad (5)$$

The quality of the generated random numbers  $U_i$  can be investigated using a number of different measures. All these measures quantify the fulfillment of the requirements that the random number must be (a) uniformly distributed and (b) uncorrelated for different time lags as requested by Eqs. 1 and 2. Fig. 3 confirms the LCG's diffusion capacity for small seeds between 1 and 200.

The uniformity of the distribution is checked here with the Kolmogorow-Smirnow test. The correlation is tested with a two-sided Z-test of the correlation. The given  $p$ -values indicate the confidence in the assumptions on a scale from 0 to 1.  $p$ -values larger than 0.10 indicate that the property under test is confirmed.

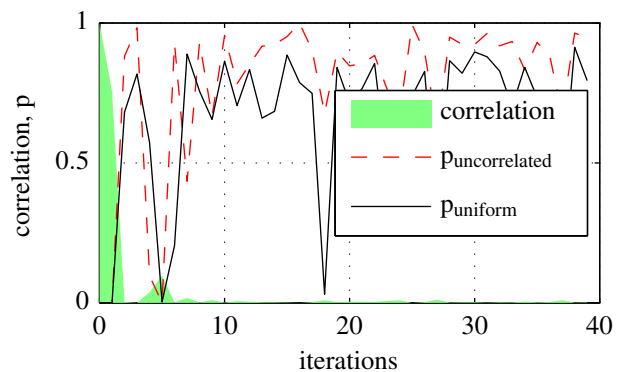


Figure 3: After ten iterations, the random numbers from an LCG can be assumed to be uniformly distributed and uncorrelated with the seed.

Table 1 gives an overview of the quality of the most important random number generators used in this work. They are compared to two standard solutions from the Modelica\_LinearSystems2 library [3] and the Design.Experimentation library [4]. It can be seen that all generators produce uncorrelated uniform random numbers.

Table 1: All RNGs produce uniform and uncorrelated random numbers.

Generator	Uniform	Uncorrelated
WELL1024a	$p = 0.396$	$p = 0.405$
LCG	$p = 0.456$	$p = 0.253$
DIRCS	$p = 0.432$	$p = 0.523$
LinearSystems2	$p = 0.508$	$p = 0.373$
Design	$p = 0.305$	$p = 0.899$

### 3.2 Probability Density Functions

Each uniform random number  $U_i$  generated with the methods presented in the previous section must be transformed to match the desired noise characteristics. The first step is to apply a mapping

$$U_i \mapsto X_i \tag{6}$$

according to a probability density function (PDF)  $f(x)$ . This yields random numbers  $X_i$  with a specified distribution. Informally, the function  $f(x)$  describes the probability that a random variable  $X_i$  equals  $x$ .

There are several methods to achieve this step (see e.g. [5]). If an analytic PDF is given, then often its primitive  $F(x) = \int_{-\infty}^x f(\tilde{x})d\tilde{x}$  can be derived. This cumulative density function (CDF) describes the probability that the random variable  $X_i$  is smaller or equal to  $x$ . Using the inverse of the CDF, a random number with the given distribution can be analytically calculated by  $X_i = F^{-1}(U_i)$ . This method is illustrated below for the heavy-tailed Cauchy-Lorentz distribution with the location parameter  $\mu$  and the scale parameter  $\gamma$ .

$$f_{\text{Cauchy-Lorentz}}(x) = \frac{1}{\pi} \left( \frac{\gamma}{(x - \mu)^2 + \gamma^2} \right) \tag{7}$$

$$F_{\text{Cauchy-Lorentz}}(x) = \frac{1}{\pi} \arctan \left( \frac{x - \mu}{\gamma} \right) + \frac{1}{2} \tag{8}$$

$$F_{\text{Cauchy-Lorentz}}^{-1}(U_i) = \gamma \tan \left( \pi \left( U_i - \frac{1}{2} \right) \right) + \mu \tag{9}$$

$$= X_{i,\text{Cauchy-Lorentz}} \tag{10}$$

If an analytical inverse of the CDF cannot be derived, different methods have to be employed. A prominent example is the normal distribution, which does not have an analytical CDF. To generate a normally distributed random variable, we employ the Box-Muller transform [6]. It uses two uniform random numbers  $U_{1i}$  and  $U_{2i}$  to calculate  $X_i$  according to

$$X_{i,\text{normal}} = \mu + \sigma \sqrt{-2 \ln U_{1i}} \cos(2\pi U_{2i}). \tag{11}$$

Finally, if no direct transformation is given in the literature, some common distributions can also be calculated directly according to their definition. The Bates distribution e.g. describes the distribution of an average over  $n$  uniform random numbers and can be calculated directly from

$$X_{i,\text{Bates}} = \frac{1}{n} \sum_{k=1}^n U_{k-i}. \tag{12}$$

### 3.3 Power Spectral Densities

In the previous sections, we have shown how to generate a discontinuous sequence of random variables  $X_i$  at an arbitrary sampling rate  $\Delta t$ . This sequence has to be processed further, in order to shape a continuous signal  $r(t)$  with a specified frequency content. A common method is to apply a simple low-pass filter to the noise sequence. Although being common practice, this method has two main disadvantages:

1. The simulation speed is limited by the high sampling frequency of the noise signal, the small time-constant of the low-pass filter or both.
2. It is often unclear which statistical properties the filtered signal has. It often differs widely from the PDF of the discrete signal.

For many applications it is thus favorable to compute the continuous signal directly out of the discrete sequence without using dynamic states as in usual implementations of low-pass filters. In this work, we continuously interpolate the discrete sequence using kernel functions.

If a kernel function  $K(t)$  is given, then the interpolation can be expressed as a linear combination of the kernel function with different weights  $X_i$  and offset  $i\Delta t$ :

$$r(t) = \sum_i X_i \cdot K(t - i\Delta t). \tag{13}$$

In order for this sum to be a proper interpolation, the kernel must equal 1 at the origin and 0 at all multiples of  $\Delta t$ :

$$K(i\Delta t) \stackrel{!}{=} \begin{cases} 1 & \text{for } i = 0 \text{ and} \\ 0 & \text{for } i \neq 0. \end{cases} \tag{14}$$

A prominent kernel function, which fulfills these constraints, is the *hat* function used to create a linear interpolation.

$$K_{\text{hat}}(t) = \begin{cases} 1 - \left| \frac{t}{\Delta t} \right| & \text{if } t \in [-\Delta t, +\Delta t] \\ 0 & \text{else} \end{cases} \tag{15}$$



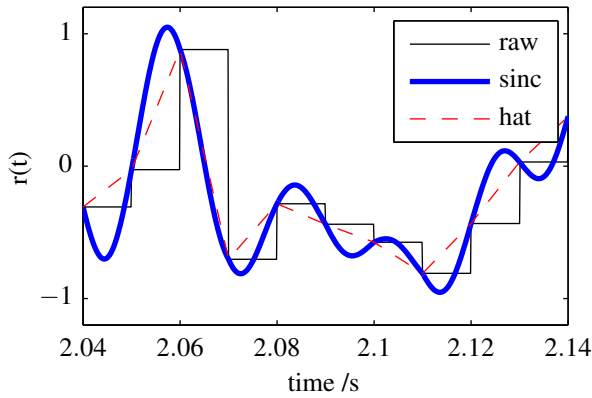


Figure 4: Different interpolations can be applied to the random sequence. The sinc interpolation achieves a very smooth signal  $r(t)$ .

The resulting signal  $r(t)$  is a linear combination of the kernel functions. Its frequency content is thus fully determined by the frequency content of the kernel function. If the sum is truncated by limiting the number of involved sampling points  $X_i$ , additional content is introduced by the discontinuous support. In practice, however, the truncated contributions are often negligible, leading to acceptable approximations. This especially holds true for the hat function, which can be well truncated to only include two random samples.

Another important kernel function is the *normalized sinc* function:

$$K_{\text{sinc}}(t) = 2B \text{sinc}(2Bt) = \frac{\sin(2B\pi t)}{\pi t}. \quad (16)$$

It only contains frequencies up to its bandwidth  $B$ . If  $B = 1/(2\Delta t)$  is chosen to match half the sampling rate, the normalized sinc function also fulfills the constraints of an interpolation kernel. Interpolation with the sinc function can thus be used to apply an optimal low-pass filter to the random sequence.

Figure 4 shows the different interpolation methods described above. The interpolation is applied to a random sequence  $X_i$  generated with 100 S/s. The frequency contents of the signals are compared in Fig. 5. The raw sample-and-hold signal of  $X_i$  contains frequencies higher than the sampling frequency. Using the interpolation with the sinc function, the frequency characteristic can be nicely limited to the desired cut-off frequency of half the sampling frequency. It is important to understand how the interpolation affects the statistical properties of the random signal  $r(t)$ . Since  $r(t)$  is a weighted sum of statistically independent random variables, we can use the following two laws to

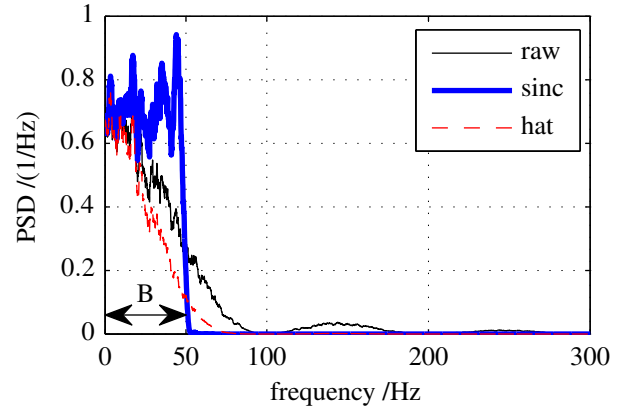


Figure 5: The frequency characteristic are shaped by the interpolation function. The sinc function achieves a very good low-pass characteristic.

determine the change of variance for any valid filter:

$$\text{Var}(X_i \cdot c) = \text{Var}(X_i) \cdot c^2, \quad (17)$$

$$\text{Var}(X_i + X_j) = \text{Var}(X_i) + \text{Var}(X_j). \quad (18)$$

Here,  $X_i$  and  $X_j$  denote the statistically independent random variables generated by the RNG and  $c$  denotes a constant weight. The variance of the random variables is fixed by the selected PDF and the constant  $c$  is given by the interpolation kernel. The time-dependent variance of  $r(t)$  from Eq. 13 can thus be expressed as

$$\text{Var}(r(t)) = \text{Var}(X_i) \cdot \sum_i (K(t - i\Delta t))^2. \quad (19)$$

Due to the constraints for interpolation kernels Eq. (14), the variance of  $r(t)$  at the sampling points is equal to the variance of the random variable  $X_i$ . In between the sampling points, the variance of the random signal is a function of the time. We can compute the expectation value of the variance for the entire signal by formulating the integral over the interval  $\Delta t$ :

$$\begin{aligned} E[\text{Var}(r(t))] &= \frac{1}{\Delta t} \int_{t=0}^{\Delta t} \text{Var}(r(t)) dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} \int_{t=0}^{\Delta t} \sum_{i=-n+1}^n (K(t - i\Delta t))^2 dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} \sum_{i=-n+1}^n \int_{t=0}^{\Delta t} (K(t - i\Delta t))^2 dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} \int_{t=-n\Delta t}^{n\Delta t} K(t)^2 dt. \end{aligned} \quad (20)$$

Doing this computation for the linear interpolation using the hat function leads to

$$E[\text{Var}(r(t))] = \frac{2}{3} \text{Var}(X_i). \quad (21)$$

Thus, any linear interpolation reduces the variance of the input signal by one third, independently of the PDF used to generate  $X_i$ . In a similar manner, the result can be computed for the optimal bandwidth limitation. Fortunately,

$$\lim_{n \rightarrow \infty} \frac{1}{\Delta t} \int_{t=-n\Delta t}^{n\Delta t} K_{\text{sinc}}(t)^2 dt = 1 \quad (22)$$

and hence the variance is only minorly affected for  $n$  chosen large enough. In fact, with  $n = 3$ , the expected variance is only changed by less than 5 %.

Not only the variance may be affected but also the shape of the PDF. It is possible to compute this effect for specific PDFs but the most general statement can be drawn from the central limit theorem. It states that a sum of many independent random variables (with finite variance) is approximately normally distributed. Hence we can state that the application of a filter transforms all PDFs with finite variance gradually to look like the normal distribution and that this effect is expected to be stronger the wider the domain of  $K(t)$  is.

Our analysis suggests a very suitable combination for generating a continuous random noise signal: if the discrete noise is generated by a normal distribution and if it is interpolated by an ideal bandwidth limitation, the resulting signal is also of a normal distribution with the same variance and a frequency characteristic below the cut-off frequency that represents white noise. The cut-off frequency will be half the sampling frequency.

## 4 Implementation in Modelica

The concepts laid out in Sec. 3 are implemented in the Modelica Noise library. An overview of the library's components is shown in Fig. 6. The library provides a single block PRNG as the interface to all of its facilities. This block is described here.

In order to provide the user with a convenient interface, the PRNG block combines all parts of the noise generation process described in Sec. 3. The block's parameter pane is shown in Fig. 8.

All three parts of the noise generation can be (almost) independently parametrized. The *Random Number Generator (RNG)* parameters allow the user to select whether a sampling-based or a sampling-free generator shall be used. Two selectors allow to maintain parametrized instances of both variants in the PRNG block. The parameters of the generators can all be set via the PRNG's parameter interface. The criteria listed in Table 1 may assist in choosing a suitable RNG.

The *Probability Density Function (PDF)* can also be selected in the PRNG's parameter pane along with its parameters. Some of the available distributions are shown

in Fig. 7. Cauchy-Lorentz, Irwin-Hall, Bates and Discrete distributions are also available. The distribution should be selected according to the specification of the desired noise. Additional distributions can be implemented according to the literature by filling a common function interface.

Finally, a kernel function for the desired *Power Spectral Density (PSD)* can be selected and parametrized. The kernels are set up with standard parameters to match the update rate of the block. For example, the ideal low-pass filter is set up with a cut-off frequency of half the update rate. The implemented filters are shown in Fig. 9. A typical choice of the PSD would be to use the raw random sequence or the ideal low-pass interpolation with  $n \geq 5$ . Additional kernels can be added easily by the user.

The (*Pseudo-*) *Sampling* parameters of the PRNG block are the same as for most sampling Modelica blocks. A `startTime` determines the first sample to be generated. A `samplePeriod` determines the step-size of the sampler. For sample-free generators, the sampling is relaxed to a pseudo-sampling of the random number, i.e. imposing an upper limit to the update rate. An additional `infiniteFreq` switch can be used to completely disable (pseudo-)sampling and let the simulation tool handle the step-size of the generator based on the desired integration accuracy. This also disables the PSD filtering, because the interpolation needs discrete noise samples.

The *Enable/Disable* parameters finally are used to enable noise generation or to output a simple dummy variable `y_off`. It can be used to parametrically restore the ideal system without noise.

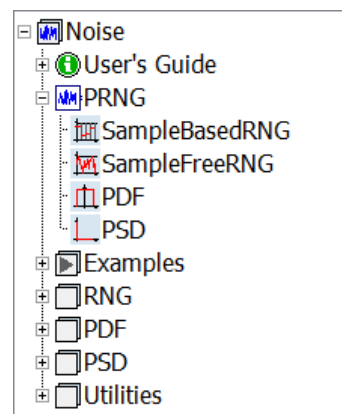


Figure 6: The Noise library provides a ready-to-use noise block as well as a collection of RNG, PDF and PSD implementations.

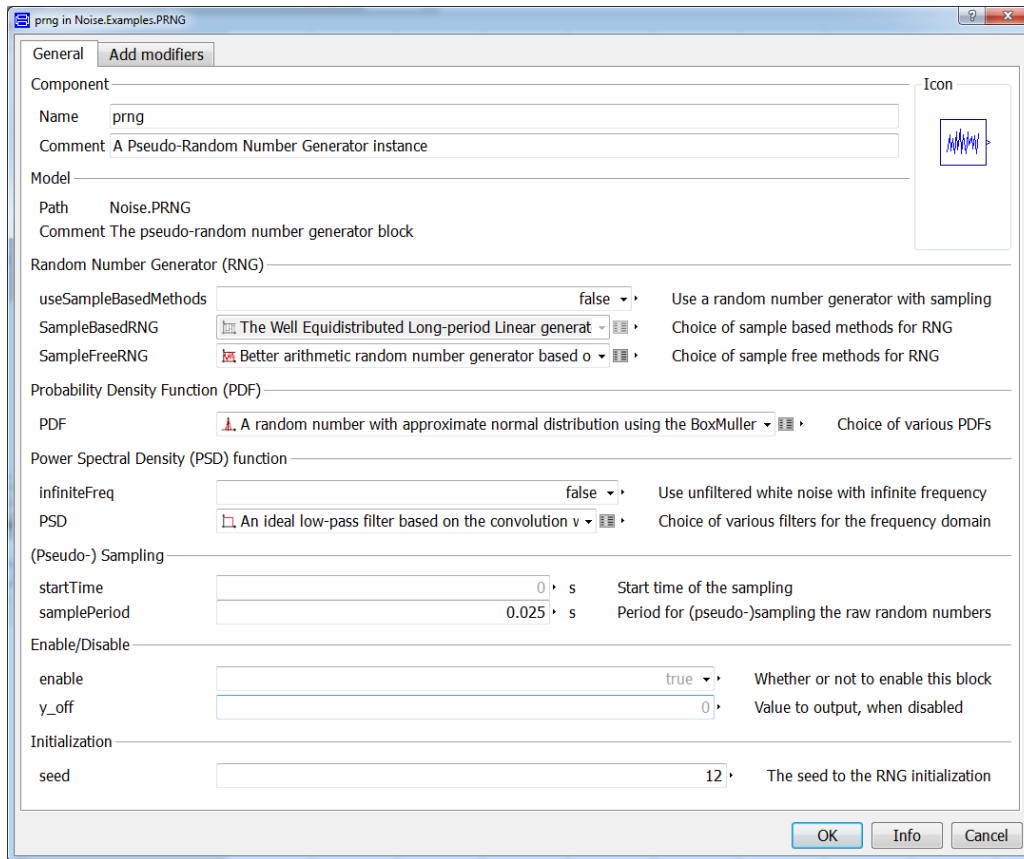


Figure 8: The PRNG block can be used to collectively set up custom noise by modularly combining all available RNG, PDF and PSD implementations. Additional switches are provided to set the samplePeriod, enable sample-free RNGs and enable infinite frequency emulation.

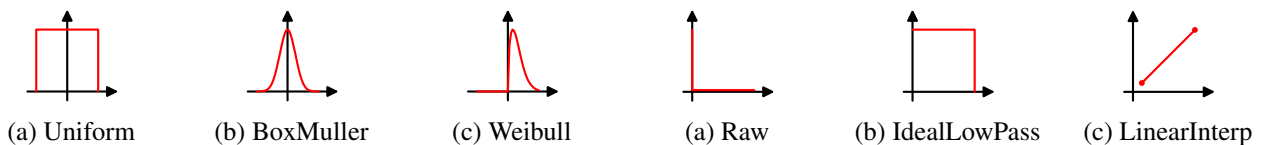


Figure 7: Some of the provided PDFs are a uniform, a normal and a Weibull distribution.

Figure 9: The provided PSDs include an unfiltered white noise, an ideal low-pass filter and a linear interpolation.

## 5 Evaluation of Filtered Noise

Relevant noise in realistic applications is in most cases sampled and filtered (see e.g. Fig. 2). The process of sampling and filtering changes the probability density as well as the frequency content of the signal. In the following sections, we will show how these characteristics can be modeled using the Noise library. First, a synthetic example is used.

In this section, the probability distribution and frequency content of noise generated with a standard approach and with the Noise library are compared. A digital sensor is used as an example. The sensor has a uniform noise distribution with amplitudes between  $-0.05$  rad and  $0.05$  rad. The signal is sampled with

6 kS/s. The signal is subsequently smoothed with a running mean filter using 20 samples and then down-sampled by a factor of 20. This procedure is representative for a typical angular resolver signal used for control purposes. In order to introduce a model of a simple system, a CriticalDamping block from the Modelica standard library is used. It has a fixed cut-off frequency of 10 Hz and a variable number of states.

### 5.1 Model using the LinearSystems library

In Fig. 10a, the reference implementation using the Modelica\_LinearSystems library is shown. Uniform noise is generated with at a rate of 6 kS/s, filtered with a running mean FIR filter and then down-sampled.

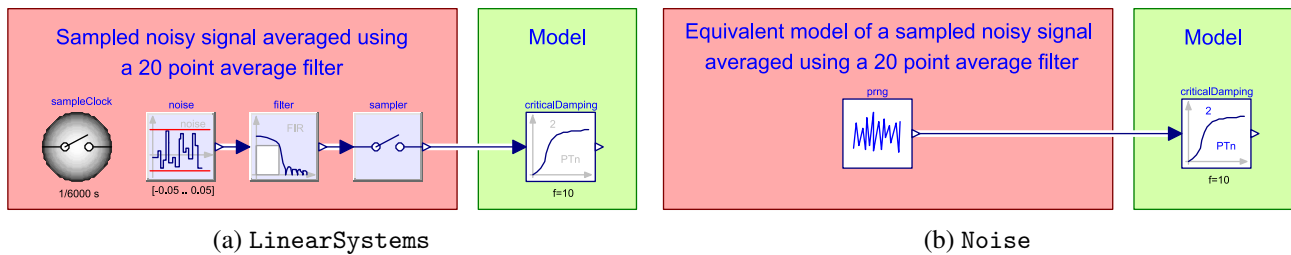


Figure 10: Generation of realistic filtered noise using the LinearSystems and the new Noise library.

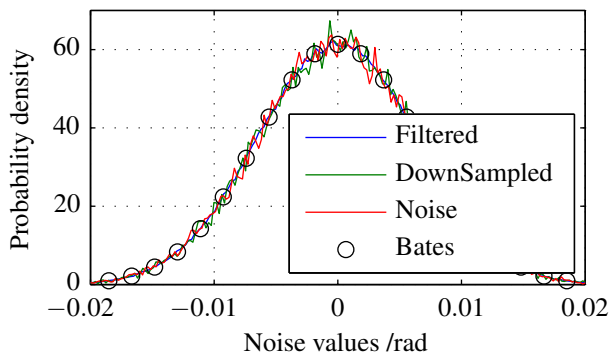


Figure 11: Histograms of the different noise signals

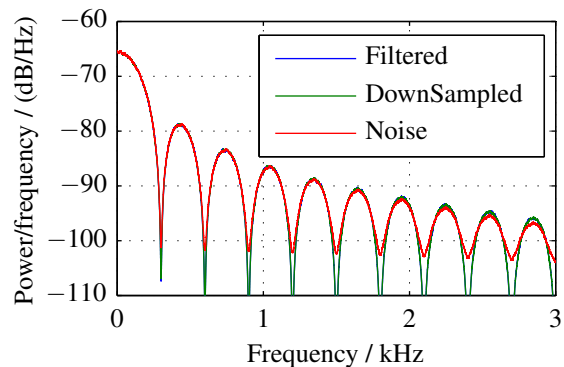


Figure 12: Frequency analysis of the noise signals.

### 5.2 Model using the Noise library

Using the Noise library, it is possible to generate the filtered and down-sampled noise signal without sampling at a high frequency. In order to achieve an equivalent signal, the PDF must be adjusted to match the distribution of the filtered signal. This is done selecting the Bates distribution described in Sec. 3.2 with  $n = 20$ . The noise signal can then be generated at the down-sampled update rate of 300 S/s. Additionally, the DIRCS generator is used to suppress all time-events in the simulation. The Modelica block diagram is shown in Fig. 10b.

### 5.3 Probability density distribution

The empirical probability density functions of the generated noise signals are shown in Fig. 11. To show that down-sampling the signal has no influence on the PDF, the down-sampled, as well as the original signals are shown. The probability density function of the generated noise signal is obtained experimentally by simulating both noise models for 200 s and by using 250 bins between the minimal and maximal values (-0.05 to 0.05). The results of the analysis show a good match between the different signals.

### 5.4 Frequency content

The frequency content of the signal is an important property of a noisy signal. To show that also the

frequency content of the PRNG block mimics the frequency content of the sampled and filtered signal Welch’s power spectral density analysis is applied to the signals (see Sec. 2). The results are shown in Figure 12. The plot shows that the frequency contents of all signals correspond very well. Especially the low frequency content up till 300 Hz is a very good match.

### 5.5 Simulation times

In order to compare the simulation times of the presented models, the models are simulated again for 200 s. As a reference, a standard noise block from the LinearSystems library without downsampling is included using 300 Samples/s, the same update rate as the PRNG block updates internally. Only 10 output points are generated in order to make sure the output routine does not influence the simulation times. The simulations are performed using Dymola 2014 FD01 Beta 3 on an Intel® Xeon® E5-1620 processor. The results of the simulations are summarized in Table 2. Additionally to the model structure, the number of states in the CriticalDamping system is varied.

The results clearly show the advantage of using the Noise library for modeling noise in a system. The simulation time for the Noise model with a system of second order is six times faster than the standard approach. If the complexity of the system is increased by using a number of 50 states, the Noise model is 26 times faster

Table 2: Simulation times and number of steps of the models with noise for 200 seconds simulation. LS 6kS/s filtered marks the Noise from Figure 10a, LS 300S/s raw marks the simulation results of the noise generation using the LinearSystems library using a direct noise output of the signal and LS 300S/s Bates using a Bates distribution using an averaging of 20 random samples at each timestep.

Model	States	Time	Events
LS 6kS/s filtered	2	45 s	12e6
LS 300S/s raw	2	6 s	60e3
LS 300S/s Bates	2	8.7 s	60e3
Noise	2	7.5 s	0
LinearSystems	50	256 s	12e6
LS 300S/s raw	50	23 s	60e3
LS 300S/s Bates	50	24.5 s	60e3
Noise	50	9.7 s	0

than the model using LinearSystems noise. The approaches without downsampling (LS 300S/s methods) have a better performance. The Bates distribution at 300S/s has a similar calculation time as the method using the Noise library. However, at higher model levels, for a system with 50 states, the speedup ratio is a little over 2. This result show that integrator restarting due to the event generation becomes more expensive at increasing system complexity.

## 6 Industrial Application Example

In order to test the performance of the presented noise models in an application of industrial complexity, the example of an electro mechanical actuator is chosen. This actuator is generated using the Actuator toolbox

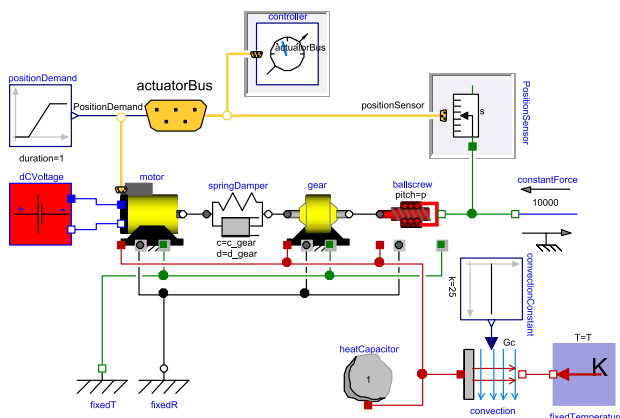


Figure 13: Actuator model overview. The motor position sensor is used to include the noise effects.

[7]. In Figure 13, an overview of the model is given.

The position sensor of the motor can be chosen from an ideal sensor or two sensor versions with noise. The sensor readings are used for controlling the motor current as well as the speed and position of the actuator. To obtain the motor speed, position is differentiated. For these reasons, adding noise to the system is expected to strongly influence the system's behavior.

To test the presented noise generation and compare it with a traditional approach, two sensors with noise were derived. The first sensor uses a traditional sampled noise using the blocks from Modelica\_LinearSystems.Sampled, the second uses the Noise library presented in this work. The two noise models are the same as presented in Sec. 5. The noise is assumed to be additive as shown in Fig. 1.

### 6.1 Simulation results

The actuator model is simulated with each of the three sensor versions. The actuator is commanded to follow a change in position of 43 mm at  $t = 1$  s. Figure 14 shows the responses with the three sensor versions. The results match very well. The simulations were carried out using the Dassl integrator with a tolerance of  $10^{-4}$ . For the following comparison, only 10 output intervals are requested in order not to influence the simulation times by the output intervals. The simulation times and generated events are summarized in Table 3.

### 6.2 Time and state events

One of the main benefits of the presented noise model is that no time events are generated. As expected, the sampled noise using the LinearSystems library generates 15000 time events, which is the product of the sample rate and the simulation time. The model built with the Noise library, however, does not generate any time events. The amount of state events generated by

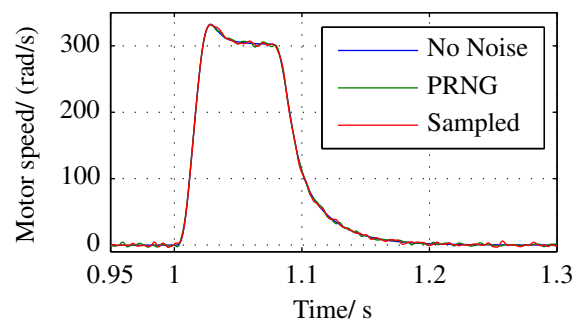


Figure 14: Resulting trajectory of the actuator using the three sensor versions. The actuator follows a commanded position change of 43 mm.

Table 3: Simulation results of the actuator models over 2.5 seconds.

Model	Time	Events	
		Time	State
No Noise	0.037 s	1	12
LinearSystems	27.5 s	15000	1642
Noise	10.5 s	1	1791

both models is roughly the same. These events are generated by the nonlinearities in the model. Mainly the inverter reacts to the high-frequency changes of the demanded current, which is generated by the noisy sensor readings.

### 6.3 Simulation speed

Especially in more complex models the penalty of an event becomes large. Restarting the integrator can use as much time as an integrator step itself. In Table 3, the simulation time of the models is shown. The noise itself has a big penalty on the simulation performance. This is expected, as the position sensor is used in all control algorithms. The simulation time using the Noise library, however, is decreased with respect to the standard implementation due to the reduced number of time events.

## 7 Conclusions and Outlook

We have shown how to properly implement a noise signal in Modelica by selecting a high-quality random number generator (RNG) and by specifying a probability density function (PDF) as well as a power spectral density (PSD) of the desired signal. In order to aid the user in this process, we have proposed a library with modular implementations of the three parts of the noise generation. The library provides a convenient interface to set all parameters. It is further possible for the user to freely implement new modules.

The proposed combination of sample-free RNGs and continuous PSDs provides for a satisfactory increase in simulation speed by a factor between 2.5 in a real world actuator example and up to 26 using an academic model.

Time events due to the noise model can be completely eliminated from the simulation by using the proposed continuous DIRCS random number generator, which relies on continuously seeding a standard RNG with a function of the time.

Further extensions of the library can be seen in taking advantage of the Modelica 3.3 function arguments.

The library could additionally be improved in modularity by introducing separate seeding functions. In this way, also the continuously seeded generator could be modularized. Some convenient features such as global seeding or seeding based on the machine time may also be addressed in future versions. Additionally, the current interpolation filter may be extended to allow for arbitrary filter functions implemented by the user.

## Acknowledgements

We thank our colleague Andreas Knoblach for his valuable comments on the present paper.

## References

- [1] Crispin W Gardiner. *Handbook of Stochastic Methods: for Physics, Chemistry and the Natural Sciences*. Springer Berlin, 2nd edition, 1985.
- [2] François Panneton, Pierre L'ecuyer, and Makoto Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006.
- [3] Marcus Baur, Martin Otter, and Bernhard Thiele. Mod- elica libraries for linear control systems. In *Proceedings of 7th International Modelica Conference, Como, Italy, September*, pages 20–22, 2009.
- [4] Dassault Systèmes AB. *Dymola User's Manual – Volume 2*, 2011.
- [5] Ralf Korn, Elke Korn, and Gerald Kroisandt. *Monte Carlo Methods and Models in Finance and Insurance*. Financial Mathematic Series. Chapman & Hall / CRC Press, 2010. ISBN 978-1-4200-7618-9.
- [6] George EP Box and Mervin E Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.
- [7] Franciscus L. J. van der Linden, Clemens Schlegel, Markus Christmann, Gergely Regula, Christopher Hill, Paulo Giangrande, Jean-Charles Maré, and Imanol Egaña. Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters. In *Proceedings of the 10th International Modelica Conference*, 2014.

# A physical solution for solving the zero-flow singularity in static thermal-hydraulics mixing models

Daniel Bouskela                      Baligh El Hefni  
EDF R&D  
6, quai Watier F-78401 Chatou Cedex, France  
daniel.bouskela@edf.fr    baligh.el-hefni@edf.fr

## Abstract

For the 0D-1D modelling of thermal-hydraulics systems, it is common practice to use static mixing models to compute the mixing specific enthalpy in fluid junctions such as mergers or splitters. However, this simplification leads to a well known singularity when the mass flow rate inside the junction goes to zero. The origin of the singularity is explained, and a rigorous physical solution is proposed to eliminate the singularity. A prototype implementation has been developed in the ThermoSysPro library for power plant modelling that illustrates the interest of the proposed solution, shows the impact on the structure of the library and enables to evaluate the computing overhead with respect to several possible variants.

*Keywords: thermal-hydraulics; mixing models; convection; diffusion; ThermoSysPro*

## 1 Introduction

When modelling thermal-hydraulics at the system level, such as power plants, it is common practice to use static equations to compute fluid quantities in mixing equipments such as mergers and splitters. This simplification stems from the fact that the volume of mixing is often neglected in junctions, therefore eliminating the differential term in the balance equations. It also occurs when computing isolated operating points that only require static models.

Neglecting diffusion is very common when one deals with large mass flow rates, as diffusion is only significant when mass flow rates approach zero. When diffusion is neglected, the only thermal phenomenon remaining in the model is convection. However when mass flows go to zero, convection disappears. So if diffusion is neglected, when mass flow rates go to zero, as convection also disappear, there is no thermal phenomena left in the model, leading to a possible indetermination of the enthalpy. This indetermination results in a singularity when

static models are used, because in such case there is no differential variable to act as a memory for the enthalpy when mass flow rates are equal to zero.

In subsequent chapters, the mathematical origin of the singularity is explained. Then a rigorous mathematical formulation is proposed based on physical insight to remove the singularity. The idea is to reintroduce diffusion in static mixing models. Finally, a performance benchmark is given, based on a prototype implementation in ThermoSysPro. ThermoSysPro is a Modelica library developed by EDF for the modelling of power plants of all types [1].

## 2 Computing the state of a thermal-hydraulics system

As the objective is to find the origin of the physical singularity before giving a solution for removing the singularity, it is useful to understand how the physical state of a thermal-hydraulic system such as a volume is defined.

A volume is an abstract physical component where incoming flows mix. Figure 1 features four incoming flows. Flows are positive when they enter the volume and negative otherwise.

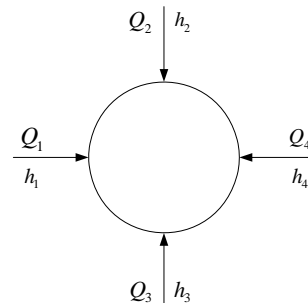


Figure 1: volume

In general, the state of a physical system is given by the set of independent physical quantities that completely define the state. There are many ways to choose the state variables for a given physical sys-

tem. For a thermal-hydraulic volume  $a$ , a common choice is to use the average pressure  $P_a$  and the average specific enthalpy  $h_a$  inside the volume. Then the state of the volume  $a$  will be defined if  $P_a$  and  $h_a$  can be computed.

In the sequel, we are only interested in computing  $h_a$  which is called the mixing enthalpy in volume  $a$ .

To compute  $h_a$ , one must consider the neighboring volumes of  $a$  which are collectively denoted  $b$  (see Figure 2).

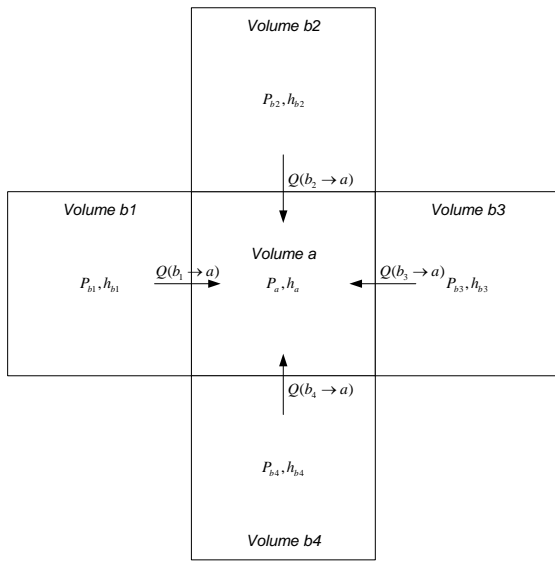


Figure 2: grid scheme

Each volume is assumed to be in thermodynamic equilibrium, so that their thermodynamic state is physically defined. However neighboring volumes may have different physical states, so that pressure and temperature gradients may exist that cause mass and energy flows between neighboring volumes through their common limiting boundary.

Mass flowing from volume  $b$  to volume  $a$  is denoted  $Q(b \rightarrow a)$ . Therefore  $Q(b \rightarrow a)$  is positive if the flow actually occurs from  $b$  to  $a$ , and negative otherwise. So  $Q(a \rightarrow b) = -Q(b \rightarrow a)$ .

Notice that the relation

$$Q(a \rightarrow b) + Q(b \rightarrow a) = 0$$

is not a mass balance equation between volumes  $b$  and  $a$ , but merely states the fact that  $Q(b \rightarrow a)$  and  $Q(a \rightarrow b)$  denote the same physical quantity with opposite sign conventions.

The specific enthalpy of flow  $Q(b \rightarrow a)$  is denoted  $h_{b:a}$ . The meaning of notations is recalled in chapter 7.

The dynamic mass and energy balance equations are given by

$$\frac{d(\rho_a \cdot V_a)}{dt} = \sum_b Q(b \rightarrow a)$$

$$\frac{d(\rho_a \cdot V_a \cdot u_a)}{dt} = \sum_b (h_{b:a} \cdot Q(b \rightarrow a) + J(b \rightarrow a)) + W_a$$

$J(b \rightarrow a)$  is the energy flow through diffusion.

$$J(b \rightarrow a) = -A_{b:a} \cdot k_{b:a} \cdot \nabla T(b \rightarrow a) \quad (1)$$

The static mass and energy balance equations are obtained by eliminating the dynamic terms on the left hand sides.

$$0 = \sum_b Q(b \rightarrow a) \quad (2)$$

$$0 = \sum_b (h_{b:a} \cdot Q(b \rightarrow a) + J(b \rightarrow a)) + W_a \quad (3)$$

As the quantity  $h_a$  does not appear explicitly in the static energy balance equation, it must be computed through the quantities  $h_{b:a}$ . So the relation between  $h_a$  and  $h_{b:a}$  must be established.

To that end, the fluid vein between volumes  $a$  and  $b$  is considered (see Figure 3).

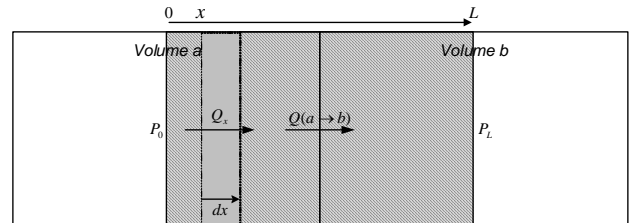


Figure 3: fluid vein

The static mass and energy balance equations in the volume limited by  $dx$  is

$$0 = \frac{\partial Q_x}{\partial x} \quad (4)$$

$$0 = \frac{\partial}{\partial x} \left( h_x \cdot Q_x - A \cdot k \cdot \frac{\partial T_x}{\partial x} + W_x \right) \quad (5)$$

Eq. (4) states that  $Q_x$  is constant. Therefore:

$$Q_x = Q(a \rightarrow b) = -Q(b \rightarrow a)$$

In order to find an analytical solution to Eq. (5), the following relation between  $dh_x$  and  $dT_x$  is considered:



$$dh_x = c_p \cdot dT_x \quad (6)$$

In the sequel, it is assumed that the relation given by Eq. (6) is valid (i.e. outside of the saturation line, for isobaric transformations, or for ideal gases, or when the contribution of the pressure variation to the variation of the specific enthalpy is negligible as compared to the contribution of the variation of temperature).

Under the additional assumption that  $W_x = 0$ , the energy balance equation writes:

$$0 = Q(a \rightarrow b) \cdot \frac{\partial h_x}{\partial x} - \frac{A \cdot k}{c_p} \cdot \frac{\partial^2 h_x}{\partial x^2} \quad (7)$$

Eq. (7) can be solved analytically [2]:

$$h_x = \frac{1}{1 - e^{P_e}} \cdot \left[ h_b - h_a \cdot e^{P_e} + (h_a - h_b) \cdot e^{P_e \cdot \frac{x}{L}} \right] \quad (8)$$

with

$$P_e = \frac{1}{\gamma} \cdot Q(a \rightarrow b) \quad (9)$$

$$\gamma = \frac{A \cdot k}{c_p \cdot L} \quad (10)$$

$h_{b,a}$  is the value of  $h_x$  for  $x = \frac{L}{2}$ :

$$h_{b,a} = h_{L/2} = \hat{s}(P_e) \cdot h_a + \hat{s}(-P_e) \cdot h_b \quad (11)$$

with

$$\hat{s}(x) = \frac{1}{1 + e^{-\frac{x}{2}}} \quad (12)$$

Figure 4 gives a plot of  $\hat{s}$ .

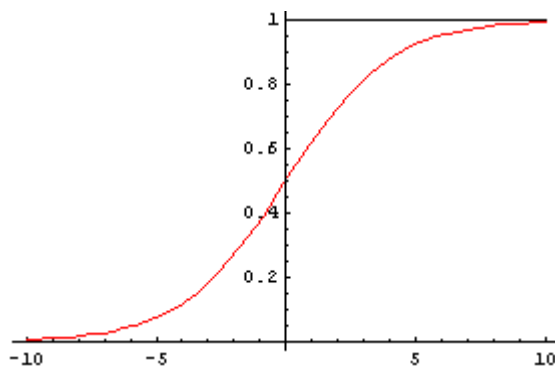


Figure 4: plot of  $\hat{s}$

If diffusion is neglected, then  $\gamma \rightarrow 0$ ,  $P_e \rightarrow +\infty$  and  $h_{b,a}$  becomes:

$$h_{b,a} = s(Q(a \rightarrow b)) \cdot h_a + s(Q(b \rightarrow a)) \cdot h_b \quad (13)$$

where  $s$  is the step function:

$$s(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (14)$$

This is the well known upwind scheme approximation for flow reversal. This relation is widely used, even if the assumptions used in this derivation are not fulfilled.

Note that  $s$  is discontinuous at  $x = 0$ , whereas  $\hat{s}$  is continuous and differentiable everywhere.

### 3 Origin of the singularity in static mixing models

The objective of this chapter is to show that the singularity in static mixing models arises when diffusion is neglected.

So in the sequel diffusion is neglected, which means that  $J(b \rightarrow a) = 0$ . Also  $Q(b \rightarrow a)$  is denoted  $Q_b$  to simplify the notation.

The mass and energy balance equations become

$$0 = \sum_b Q_b \quad (15)$$

$$0 = \sum_b h_{b,a} \cdot Q_b \quad (16)$$

The value of the enthalpy  $h_{b,a}$  is given by the upwind scheme (see Eq. (13)):

$$h_{b,a} = s(Q_b) \cdot h_b + s(-Q_b) \cdot h_a \quad (17)$$

In the sequel, the following relations are used:

$$|x| = \text{sgn}(x) \cdot x \quad (18)$$

$$\text{sgn}(x) = s(x) - s(-x) \quad (19)$$

$$s(x) = 1 - s(-x) \quad (20)$$

where  $\text{sgn}$  is the sign function.

Then using Eq. (15), (16), (17) and (20)

$$\begin{aligned} \sum_b h_{b,a} \cdot Q_b &= \sum_b s(Q_b) \cdot Q_b \cdot h_b + h_a \cdot \sum_b s(-Q_b) \cdot Q_b \\ &= \sum_b s(Q_b) \cdot Q_b \cdot h_b + h_a \cdot \left( \sum_b Q_b - \sum_b s(Q_b) \cdot Q_b \right) \\ &= \sum_b s(Q_b) \cdot Q_b \cdot h_b - h_a \cdot \sum_b s(Q_b) \cdot Q_b = 0 \end{aligned}$$

Therefore

$$h_a = \frac{\sum_b s(Q_b) \cdot Q_b \cdot h_b}{\sum_b s(Q_b) \cdot Q_b} \quad (21)$$

when  $\sum_b s(Q_b) \cdot Q_b \neq 0$ .

To find out when this condition is satisfied, using Eq. (15), (18), (19) and (20):

$$\begin{aligned} \sum_b |Q_b| &= \sum_b \text{sgn}(Q_b) \cdot Q_b = \sum_b (s(Q_b) - s(-Q_b)) \cdot Q_b \\ &= \sum_b s(Q_b) \cdot Q_b - \sum_b s(-Q_b) \cdot Q_b \\ &= \sum_b s(Q_b) \cdot Q_b - \sum_b (1 - s(Q_b)) \cdot Q_b \\ &= 2 * \sum_b s(Q_b) \cdot Q_b - \sum_b Q_b = 2 * \sum_b s(Q_b) \cdot Q_b \end{aligned}$$

Hence:

$$h_a = 2 \cdot \frac{\sum_b s(Q_b) \cdot Q_b \cdot h_b}{\sum_b |Q_b|} \quad (22)$$

when  $\sum_b |Q_b| \neq 0$ .

So when all mass flow rates are equal to zero, the mixing enthalpy  $h_a$  is indeterminate ( $h_a = 0/0$ ).

Although the indetermination occurs only at an isolated point (all mass flow rates equal to zero), it is not obvious to extend  $h_a$  in order to remove the singularity at zero (contrary to other functions with isolated singularities such as  $\sin(x)/x$ ).

In particular, it is not sufficient to replace  $s$  by  $\hat{s}$  (or in other words get rid of the upwind scheme by introducing diffusion in the flow reversing formula given by Eq. (17)) because then

$$h_a = \frac{\sum_b \hat{s}(\hat{Q}_b) \cdot Q_b \cdot h_b}{\sum_b \hat{s}(\hat{Q}_b) \cdot Q_b} \quad (23)$$

with

$$\hat{Q}_b = \frac{Q_b}{\gamma_{b,a}} \quad (24)$$

The singularity still remains since  $\sum_b \hat{s}(\hat{Q}_b) \cdot Q_b = 0$

when all mass flow rates are equal to zero.

However, noticing that

$$s(Q_b) \cdot Q_b = \max(Q_b, 0)$$

Eq. (21) may be written as

$$h_a = \frac{\sum_{Q_b > 0} Q_b \cdot h_b}{\sum_{Q_b > 0} Q_b}$$

Therefore, if one is not interested in the correct value of  $h_a$  near zero flows, which is in general the case when diffusion is neglected, then as suggested in [3] one can replace  $Q_b$  by  $\max(Q_b, Q_\epsilon)$  where  $Q_\epsilon$  is a small positive mass flow rate. Then when all mass flow rates  $Q_b$  are below  $Q_\epsilon$  ( $|Q_b| < Q_\epsilon$ ):

$$h_a = \frac{1}{N_a} \sum_b h_b \quad (25)$$

where  $N_a$  is the number of neighboring volumes  $b$  of volume  $a$ , so the singularity is removed for zero flows.

Noticing that  $\max(x, y) = s(x - y) \cdot (x - y) + y$ , one can even have a  $C^\infty$  way of removing the singularity by (1) considering the function

$$s_p(x) = \frac{1}{1 + e^{-p \cdot x}} \quad (26)$$

by (2) replacing  $s$  by  $s_p$  in the max function above

$$\text{smoothMax}(x, y, p) = s_p(x - y) \cdot (x - y) + y$$

and by (3) properly adjusting the value of  $p$  wrt.  $Q_\epsilon$ .

This solution will not be developed here any further, as a full physical solution is sought. The reason is that replacing  $Q_b$  by  $Q_\epsilon$  when  $|Q_b| < Q_\epsilon$  for computing the mixing enthalpy violates the energy balance fundamental law, so requires a proper choice of  $Q_\epsilon$  wrt. the problem at hand.

As suggested in [3] there are also other ways to write *smoothMax*.

Notice also that  $\hat{s} = s_{0.5}$  and  $s = s_\infty$  (see Eq. (12), (14) and (26)).

## 4 Removing the singularity at zero flows

Diffusion is reinstated in the energy balance equation. Then

$$0 = \sum_b Q(b \rightarrow a) \quad (27)$$

$$0 = \sum_b (h_{b,a} \cdot Q(b \rightarrow a) + J(b \rightarrow a)) \quad (28)$$

assuming without loss of generality that  $W_a = 0$ .

Using Eq. (1) and (6):

$$J(b \rightarrow a) = (A \cdot k)_{b,a} \cdot \left( \frac{\partial T_x}{\partial x} \right)_{L/2}$$

$$= \left( \frac{A \cdot k}{c_p} \right)_{b,a} \cdot \left( \frac{\partial h_x}{\partial x} \right)_{L/2}$$

Taking the derivative of Eq. (8) wrt.  $x$  at  $x = \frac{L}{2}$  yields:

$$J(b \rightarrow a) = r(P_e) \cdot J_0(b \rightarrow a) \quad (29)$$

with

$$J_0(b \rightarrow a) = \gamma_{b,a} \cdot (h_b - h_a) \quad (30)$$

$$\gamma_{b,a} = \left( \frac{A \cdot k}{c_p \cdot L} \right)_{b,a} \quad (31)$$

$$P_e = -\frac{1}{\gamma_{b,a}} \cdot Q(b \rightarrow a) \quad (32)$$

$$r(x) = \begin{cases} \frac{x}{e^{\frac{x}{2}} - e^{-\frac{x}{2}}} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases} \quad (33)$$

$J_0(b \rightarrow a)$  is the energy flux when  $Q(b \rightarrow a) = 0$ .

For easier computation  $r(x)$  may be approximated by the Gaussian

$$\hat{r}(x) = e^{-0.033 \cdot x^2} \quad (34)$$

The plot below compares  $r$  in red with  $\hat{r}$  in blue.

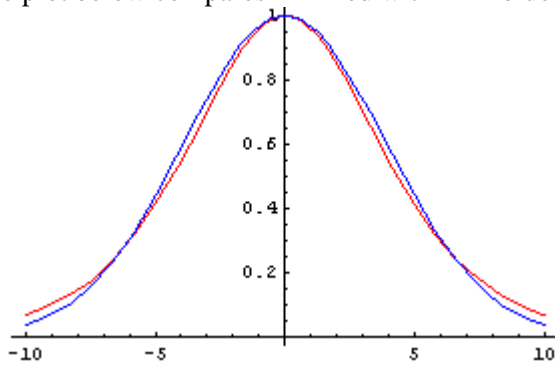


Figure 5: plot of  $\hat{r}$  and  $r$

When all flows  $Q_b$  are equal to zero, the energy balance equation writes

$$0 = \sum_b J_0(b \rightarrow a) = \sum_b \gamma_{b,a} \cdot (h_b - h_a)$$

As the coefficients  $\gamma_{b,a}$  are always strictly positive ( $\gamma_{b,a} > 0$ ), when all flows are equal to zero the mixing enthalpy  $h_a$  is defined and takes the value:

$$h_a = \frac{\sum_b \gamma_{b,a} \cdot h_b}{\sum_b \gamma_{b,a}} \quad (35)$$

If all coefficients  $\gamma_{b,a}$  are equal, then  $h_a$  is the arithmetic mean (see also Eq. (25))

$$h_a = \frac{1}{N_a} \sum_b h_b \quad (36)$$

where  $N_a$  is the number of neighboring volumes  $b$  of volume  $a$ .

As a conclusion to this chapter, when diffusion is taken into account, the energy balance equation is

$$0 = \sum_b (h_{b,a} \cdot Q(b \rightarrow a) + (h_b - h_a) \cdot Q_\varepsilon(b \rightarrow a)) \quad (37)$$

with

$$Q_\varepsilon(b \rightarrow a) = r \left( -\frac{1}{\gamma_{b,a}} \cdot Q(b \rightarrow a) \right) \cdot \gamma_{b,a} \quad (38)$$

The terms  $Q_\varepsilon(b \rightarrow a)$  are in general small but are always strictly positive and have the same physical unit as a mass flow rate (kg/s). So they never go to zero, even when all mass flow rates go to zero. They act therefore as small positive mass flow rates that remove naturally in a  $C^\infty$  way the singularity of the mixing enthalpy at zero flows.

## 5 Benchmark of the proposed solution

To evaluate the computing overhead of introducing diffusion to solve the singularity problem, the benchmark consists in comparing two alternatives for the static energy balance equation (see Eq. (28)):

- Alternative 1: without diffusion, with or without upwind scheme.
- Alternative 2: with diffusion, with or without upwind scheme.

Without diffusion means that

$$J(b \rightarrow a) = 0$$

With diffusion means that

$$J(b \rightarrow a) = \hat{r}(\hat{Q}_b) \cdot \gamma_{b,a} \cdot (h_b - h_a)$$

With upwind scheme means that

$$h_{b,a} = s(Q_b) \cdot h_b + s(-Q_b) \cdot h_a$$

Without upwind scheme means that

$$h_{b,a} = \hat{s}(\hat{Q}_b) \cdot h_b + \hat{s}(-\hat{Q}_b) \cdot h_a$$

$\hat{r}(\hat{Q}_b)$  and  $\hat{s}(\hat{Q}_b)$  are defined by Eq. (12), (24) and (34).

The equations are implemented as a prototype in the ThermoSysPro library using the scheme shown in Figure 6.

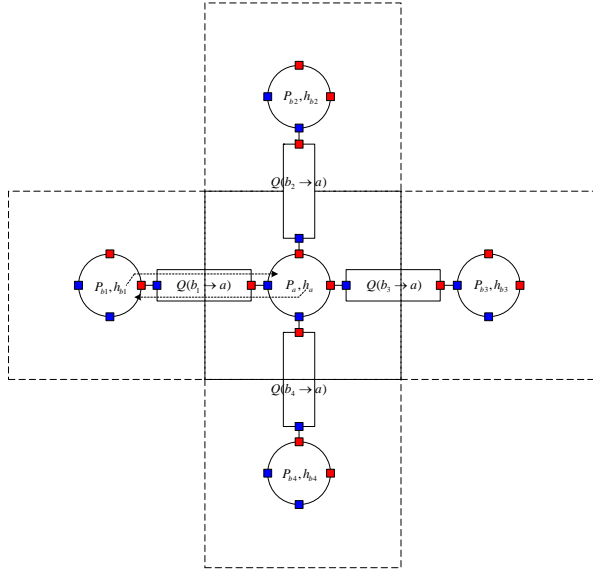


Figure 6: grid scheme in ThermoSysPro

The quantities  $(P_a, P_b)$  and  $(h_a, h_b)$  are computed in the multi-port elements with the mass and energy balance equations. Multi-port elements represent the control volumes  $a$  and  $b$ . For this reason, they are also called ‘volumes’. The quantities  $J(b \rightarrow a)$  and  $h_{b,a}$  are also computed in the volumes.

The quantities  $Q(b \rightarrow a)$  are computed in the two-port elements with the momentum balance equations. Two-port elements represent the interfaces  $b : a$  between  $a$  and  $b$ .

The interface  $b : a$  is oriented positively from the blue port to the red port of the two-port element that represents  $b : a$ . So the mass flow rate is positive when the fluid flows along the positive direction of the interface orientation, i.e. from the blue port to the red port. To reflect this sign convention for mass flow rates, the blue port is called ‘input port’, and the red port is called ‘output port’.

The components are connected together via the input and output ports that correspond to Modelica connectors. Input and output connectors have the same structure. In order to handle diffusion, they are somewhat different from the usual fluid connectors used in Modelica fluid libraries, and in particular in the current distribution of the ThermoSysPro library. The meaning of the variables in the connector de-

pends on whether the connector is attached to a volume or to a two-port element.

If the connector is attached to a two-port element representing the interface  $b : a$  :

P	Pressure $P_a$ in volume $a$ , or pressure $P_b$ in volume $b$ , depending on whether the connector is on the side of $a$ or on the side of $b$ .
Q	Mass flow rate $Q(b \rightarrow a)$ of the fluid going through interface $b : a$ .
h	Specific enthalpy $h_{b,a}$ of the fluid going through interface $b : a$ .
h_vol_1	Specific enthalpy $h_b$ or $h_a$ of the fluid in volume $b$ or $a$ located on the side of the input port of the two-port element that represents $b : a$ .
h_vol_2	Specific enthalpy $h_b$ or $h_a$ of the fluid in volume $b$ or $a$ located on the side of the output port of the two-port element that represents $b : a$ .

If the connector is attached to a multi-port element that represents volume  $b$ , and the connector is connected to the two-port element that represents interface  $b : a$  :

P	Pressure $P_b$ of the fluid in volume $b$ .
Q	Mass flow rate $Q(b \rightarrow a)$ of the fluid going through interface $b : a$ .
h	Specific enthalpy $h_{b,a}$ of the fluid going through interface $b : a$ .
h_vol_1	If the connector is an input port: specific enthalpy $h_b$ of the fluid in the neighboring volume $b$ located in the direction of negative flow rates. If the connector is an output port: specific enthalpy $h_a$ of the fluid in volume $a$ .
h_vol_2	If the connector is an input port: specific enthalpy $h_a$ of the fluid in volume $a$ . If the connector is an output port: specific enthalpy $h_b$ of the fluid in the neighboring volume $b$ located in the direction of positive flow rates.

The purpose of  $h_{vol\_1}$  and  $h_{vol\_2}$  is to provide both  $h_a$  and  $h_b$  to volumes  $a$  and  $b$  even if they are separated by a line of connected two-port elements.

When connecting together two connectors, the variables inside the connectors are made equal because they represent the same physical quantities. So connectors are used to assemble the model from the different components, and not to generate extra physical equations (such as balance equations for instance).

This scheme for distributing the equations between multi-port and two-port elements and connecting them together enables to connect together several two-port elements without having to separate them by volumes. The connected line of two-port elements is then equivalent to a single two-port element. Also, there are no infinitesimally small volume elements implied between two connected two-port elements, so the connections do not generate the kind of singularity dealt with in this paper.

The test model is shown in Figure 7.

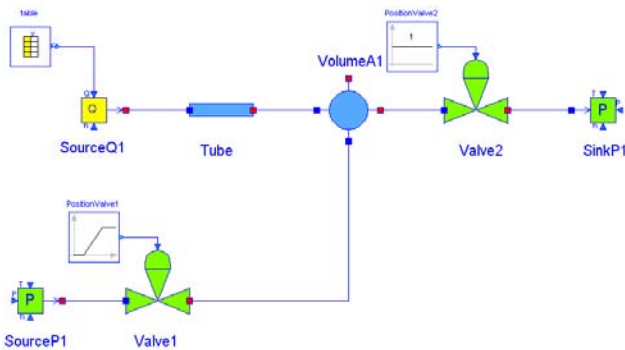


Figure 7: test model

A mixing volume (VolumeA1) is connected to two fluid sources (SourceQ1, SourceP1) and a fluid sink (SinkP1) via a pipe (Tube) and two control valves (Valve1 and Valve2). The test scenario consists in performing a flow reversal, and then setting all mass flow rates to zero.

SourceQ1	The specific enthalpy is constant equal to 1.e5 J/kg.  The mass flow rate follows the following curve (kg/s vs. s).
----------	---

SourceP1	The specific enthalpy is constant equal to 1.e5 J/kg.  The pressure is constant equal to 3 bars.
SinkP1	The temperature is constant equal to 320 K  The pressure is constant equal to 1 bar.
Valve1	The position varies from 100% to 0% in 2 seconds starting from t = 1 s.
Valve2	The position is constant equal to 100%

Four simulation runs are performed:

- Run 1.1: without diffusion, with upwind scheme
- Run 1.2: without diffusion, without upwind scheme
- Run 2.1: with diffusion, with upwind scheme
- Run 2.2: with diffusion, without upwind scheme

For each run are plotted:

- The mass flow rates at each connected port of the mixing volume (3 curves)
- The specific enthalpy inside the mixing volume (1 curve)
- The specific enthalpies inside each source and sink (3 curves).

Figure 8 gives the mass flow rates for all runs (no difference in results for all runs).

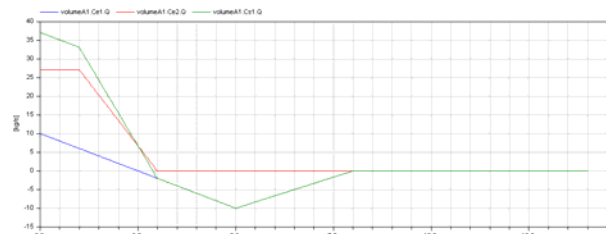
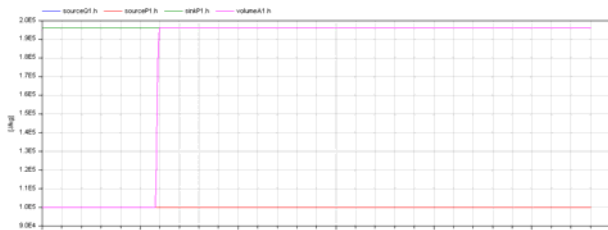


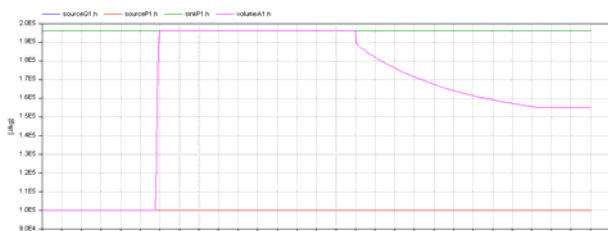
Figure 8: mass flow rates for runs 1 and 2

**Runs 1.1 and 1.2**

Figure 9 gives the specific enthalpies for run 1.1, and Figure 10 gives the specific enthalpies for run 1.2.



**Figure 9: specific enthalpies for run 1.1**



**Figure 10: specific enthalpies for run 1.2**

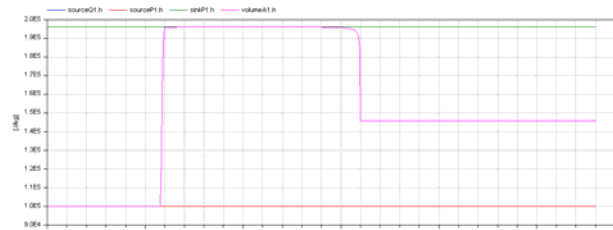
For run 1.1, when all mass flow rates are set to zero (at  $t = 8$  s), the specific enthalpy in the mixing volume keeps its last value prior to the zero mass flow rates condition, just as though there were some kind of memory holding this value when all mass flow rates become zero. This is probably an artifact due to the numerical methods used to solve the algebraic equations. The result is physically correct, but this looks as sheer luck as the theory predicts that the result is in fact mathematically undefined when diffusion is neglected.

To the contrary, for run 1.2, when all mass flow rates are set to zero (at  $t = 8$  s), the specific enthalpy in the mixing volume continues to vary until it takes a seemingly final constant value. This is a false transient which is of course unphysical because, since the model is static, all values should stay constant when the boundary conditions are constant (after  $t = 8$  s).

In both cases, the theory predicts that the mixing enthalpy can take any value when all mass flow rates are zero and diffusion is neglected, so the result is consistent with the theory.

**Runs 2.1 and 2.2**

Figure 11 gives the specific enthalpies for runs 2.1 and 2.2 (no difference in results for both runs).



**Figure 11: specific enthalpies for runs 2.1 and 2.2**

When all mass flow rates are set to zero, the specific enthalpy in the mixing volume takes the value that corresponds to the thermal equilibrium between the mixing and the sources and sink it is connected with, which is a correct physical result. The transition to thermal equilibrium is sharp but continuous.

The following table gives the computing times in seconds and the sizes of the non-linear systems after manipulation for each run with Dymola.

Run	CPU time	Sizes of non-linear systems
1.1	0.125	{ 3, 1 }
1.2	1.22	{ 3, 1 }
2.1	0.219	{ 4 }
2.2	0.313	{ 7 }

The conclusion from this experiment is that the best solution is to take into account diffusion in the energy balance equation, but still use the upwind scheme, i.e. neglect diffusion in the flow reversal equation. The overhead over the standard approximation of neglecting diffusion everywhere is 75%.

More diverse experiments should be made in order to decide whether it is better to take into account diffusion in the flow reversal formula or not, because avoiding the upwind scheme enables to remove the discontinuity due to the use of the step function.

**6 Conclusion**

Neglecting diffusion in thermal-hydraulics systems is a common approximation when dealing with large mass flow rates, as diffusion is only significant when mass flow rates are near zero.

However, this approximation leads to undefined values for the mixing enthalpies when all mixing mass flow rates are equal to zero. This is due to the fact that convection, which is the only thermal phenomena taken into account when diffusion is neglected, vanishes when mass flow rates go to zero, so there is no physical phenomenon left to describe the thermal physical state inside the mixing volume.

A rigorous mathematical and physical solution to this problem is to reinstall diffusion in the energy balance equation. This solution indeed removes the singularity for zero flows in a continuously differentiable way, as theoretically demonstrated in this paper.

A prototype implementation has been made in the ThermoSysPro library for power plant modelling, developed by EDF. The introduction of diffusion into the library has an impact on the structure of connectors.

The prototype has been tested on a small static model that features a mixing volume connected to fluid sources. The test scenario consists in performing a flow reversal, then bringing all flows to zero. The results are consistent with the theory developed in this paper. They also show that the upwind scheme, which is the equation for computing flow reversal that neglects diffusion, can be kept, as reinstalling diffusion in the flow reversal equation as well does not make any difference in the computing results, but provokes a significant overhead in computing time. However, more numerical experiments should be made to confirm this last point.

## 7 Notations

$P_a$  : fluid pressure in volume  $a$

$h_a$  : fluid specific enthalpy in volume  $a$

$u_a$  : fluid specific internal energy in volume  $a$

$\rho_a$  : fluid density in volume  $a$

$V_a$  : volume of volume  $a$

$W_a$  : external energy brought to the volume  $a$

$Q(b \rightarrow a)$  : mass flow rate of the fluid flowing from volume  $b$  to volume  $a$

$\nabla T(b \rightarrow a)$  : temperature gradient from volume  $b$  to volume  $a$

$h_{b,a}$  : specific enthalpy of the fluid flowing from volume  $b$  to volume  $a$

$(c_p)_{b,a}$  : specific heat capacity of the fluid flowing from volume  $b$  to volume  $a$

$k_{b,a}$  : diffusion coefficient at the interface between volumes  $b$  and  $a$

$A_{b,a}$  : area of the interface between volumes  $b$  and  $a$

$L_{b,a}$  : distance between the centers of volumes  $b$  and  $a$

$G_{b,a}$  : more generally, value of quantity  $G$  at the interface between volumes  $b$  and  $a$

## 8 Acknowledgments

This work was partially supported by DGCIS within the ITEA 2 EUROSYSLIB project.

## References

- [1] El Hefni B., Bouskela D., Lebreton G., 'Dynamic modelling of a combined cycle power plant with ThermoSysPro', Modelica 2011 conference proceedings.
- [2] Patankar S.V., 'Numerical Heat Transfer and Fluid Flow', Hemisphere Publishing Corporation, Taylor & Francis, 1980.
- [3] Franke R., Casella F., Otter M., Sielemann M., Elmquist H., Mattson S.E., Olsson H., 'Stream Connectors – An extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena', Modelica 2009 conference proceedings.





# Advanced Hybrid Model for Borefield Heat Exchanger Performance Evaluation, an Implementation in Modelica

Damien Picard<sup>1</sup> Lieve Helsen<sup>1</sup>

<sup>1</sup>KU Leuven, Department of Mechanical Engineering,  
Division of Applied Mechanics and Energy Conversion  
Celestijnenlaan 300A, box 2421, BE-3001 Heverlee, Belgium

## Abstract

Accurate and computationally efficient borefield models are important components in building energy simulation programs. They have not been implemented in Modelica so far. This paper describes the implementation of an innovative approach to model borefields with arbitrary configuration having both short-term (minutes) and long-term accuracy (decades) into Modelica. A step response is calculated using a combination of a short-term response model which takes into account the transient heat transfer in the heat carrier fluid, the grout and the immediately surrounding ground, and a long-term response model which calculates the boreholes interactions. Moreover, an aggregation method is implemented to speed up the calculations. Validation shows good results and very high computational efficiency.

*Keywords: Borefield; short- and long-term; Modelica; Aggregation method;*

## 1 Introduction

Building energy simulations have gained significant importance in the last decades resulting in several dynamic simulation platforms such as EnergyPlus [1] and TRNSYS [2]. Modelica might become the next generation tool for energy system simulations in buildings and communities as is the aim of the IEA EBC Annex 60 project. To achieve this goal, libraries are developed to simulate a wide variety of energy systems in buildings. Accurate and computationally efficient borefield models have not been implemented in Modelica so far, even though they play and will play an important role in recent and future buildings.

The open-source Modelica Buildings library developed by the Lawrence Berkeley National Laboratory (LBNL, US) is the only freely available library which has a U-tube single borehole model [18]. The borehole

model is similar to the EWS model implemented in TRNSYS (type 451, [17]). The model solves the transient heat flux in the ground by discretizing the surrounding ground in several cylindrical layers up to a radius of 2 meters from the borehole center. The layer temperature at this outer radius is calculated using an approximation of the line-source theory together with superposition. This temperature is updated every week in order to avoid too intensive calculations. The heat carrier fluid (HCF) and the grout (i.e. the filling material of the borehole) are simulated dynamically but their capacities are lumped. A triangle thermal resistance network is used to describe the heat transfer into the borehole heat exchanger (BHX) (i.e. from the HCF to the borehole wall). In the vertical direction, the borehole and the surrounding ground are divided into adiabatic horizontal layers. The model is not suited for multiple borehole simulation.

The E.ON Energy Research Center (Germany) also developed a single borehole model for single U-tube and coaxial type [11]. The pipe model is connected to an axially and radially discretized cylindrical ground model. A fixed temperature boundary condition is used for the ground model. The model does not take the dynamics of the grout into account and multiple borehole simulation is not possible.

Several models are implemented in TRNSYS. The Superposition Borehole Model (SBM), developed by Hellström, gives a detailed three-dimensional model for the transient thermal process in a borefield which has been implemented into TRNSYS [16]. The model can simulate single or multiple, vertical or inclined boreholes. The dynamics of the BHX is not taken into account and the computation time is very high. The Duct Heat Storage model (DST), developed by Hellström, calculates the transient thermal process for multiple borehole configurations, uniformly positioned in a cylindrical volume. The model does not take the dynamics of the BHX into account but it is fast and

it calculates the interaction between the boreholes (it uses pre-computed  $g$ -functions obtained by the SBM). Its TRNSYS implementation (type 557) can be used together with a separate program called BORE to calculate the borehole thermal resistance depending on the flow rate and the temperature [16].

To the author's knowledge, no model has been implemented in building simulation programs so far, which (i) is able to simulate any arbitrary configuration of boreholes, (ii) allows coaxial, U-tube type or double U-tube type BHX, (iii) has short- and long-term accuracy for minute-based year-long simulations, and (iv), is numerically efficient. The aim of this paper is to propose a new model, implemented in Modelica, which meets the above mentioned requirements. No ground water flow is taken into account.

Section 2 describes the model and Section 3 handles the computation of the response function and an aggregation method to speed up the computation. Finally Section 4 and 5 validate the model and give an example including a CPU-time comparison with the existing borehole model of the Buildings library. The main conclusions are summarized in section 6.

## 2 Bore field model

The proposed model is a so-called hybrid step-response-model (HSRM). This type of model uses the borefield's temperature response to a step load input. An arbitrary load can always be approximated by a superposition of step loads. The borefield's response to the load is then calculated by superposition of the step-responses using the linearity property of the heat diffusion equation. The most famous example of HSRM for borefields is probably the  $g$ -function of Eskilson [9]. The major challenge of this approach is to obtain a HSRM which is valid for both minute-based and year-based simulations. To tackle this problem, a HSRM has been implemented. A long-term response model (LTM) is implemented in order to take into account the interaction between the boreholes and the ground temperature evolution of the surrounding ground. A short-term response model is implemented in order to describe the transient heat flux in the BHX to the surrounding ground. The two models are merged into one HSRM in order to achieve both short- and long-term accuracy.

In this section, the long-term and the short-term response models are described.

### 2.1 Long-term response model

The long-term temperature response of the borefield is calculated using the model of Javed and Claesson [12]. This model is the current state-of-the-art and it proposes a compact expression to calculate the mean temperature of the borehole wall (average over the different boreholes of the borefield and over the length of each borehole).

The model is based on the spatial superposition of finite line-sources of equal length, each representing one borehole of the borefield. The finite line-source is calculated from the convolution of a point source of constant power along the depth of the borefield. The mirror of the solution at  $z=0$  is subtracted to ensure that no heat transfer occurs between the ground and the ambient air. After several mathematical manipulations to simplify the calculation, Javed and Claesson obtain the following compact expression for the mean borehole wall temperature:

$$\bar{T}_{mbhw}(t) = \frac{q_0}{4\pi\lambda} \int_{1/\sqrt{4\alpha t}}^{\infty} \left( \sum_{i=1}^N \sum_{j=1}^N e^{-r_{i,j}^2 s^2} \right) \frac{I_{1s}(Hs)}{Hs^2} ds \quad (1)$$

where  $q_0$  is the heat flux per meter length,  $\lambda$  is the ground heat conductivity,  $\alpha$  is the ground heat diffusivity ( $\lambda/(\rho c_p)$ ),  $N$  is the number of boreholes and  $H$  is the depth of the borefield.  $I_{1s}$  is defined by Eq. 2-3 and  $r_{i,j}$  by Eq. 4:

$$I_{1s}(h) = 4 \operatorname{ierf}(h) - \operatorname{ierf}(2h) \quad (2)$$

$$\operatorname{ierf}(x) = \int_0^x \operatorname{erf}(u) du = x \operatorname{erf}(x) - \frac{1}{\sqrt{\pi}} (1 - e^{-x^2}) \quad (3)$$

where  $\operatorname{erf}$  is the error function,

$$r_{i,j} = \begin{cases} r_b & \text{if } i = j \\ \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} & \text{if } i \neq j \end{cases} \quad (4)$$

where  $r_b$  is the BHX radius and  $(x_i, y_i)$  are the spatial coordinates of the center of each borehole from an arbitrary reference point.

Eq. 1 is valid for  $t > \frac{5r_b^2}{\alpha}$ , i.e after the transient part of the heat transfer through the BHX is completed [9]. The model also makes an important approximation by assuming uniform heat flux for all boreholes. The (long-term) accuracy of the model decreases for long simulation times for configurations with non-uniform heat fluxes, e.g. densely packed rectangular grid. For

more information about this approximation, we refer to Malayappan and Spittle [14]. Finally, the analytical solution assumes a uniform initial ground temperature equal to its average value.

Eq. 1 is implemented as a Modelica function. The integral of Eq. 1 and Eq. 2 are evaluated using the adaptive Lobatto rule implemented in the `Math.Nonlinear` of the Modelica Standard Library. The error function, however, is not implemented in Modelica. The publicly available c-code of Okumura [15] is compiled using the ability of Modelica to call external code.

## 2.2 Short-term response model

The short-term response model (STM) should be able to calculate the transient thermal response of the HCF, the grout and the surrounding ground accurately for time periods ranging from minutes to  $t = \frac{5r_b^2}{\alpha}$  (typically  $< 200$  hours). The interaction between the boreholes for these short times can be neglected, therefore a single borehole model is used.

The implemented STM is able to simulate boreholes with a co-axial, single-U-tube or double-U-tubes configuration. The model can compute the step response of a single borehole or that of a set of boreholes in series. Vertical discretization is also possible in case of an initial ground temperature gradient but no vertical heat transfer is computed except through the HCF. The main STM elements are the HCF, the pipes, the grout, the surrounding ground and the undisturbed ground temperature. Fig. 1 illustrates the model structure for a set of single-U-tube boreholes in series.

The dynamics of the HCF is calculated using the Fluid base classes of the open-source Buildings library [18] (`PartialFourPortInterface`, `PartialTwoPortInterface`, `TwoPortFlowResistanceParameters`, `LumpedVolumeDeclarations`) and the Media library from the Modelica Standard Library. The convection resistance between the HCF and the pipe is calculated by the correlation for smooth pipe in turbulent flow regime of Dittus-Boelter in the case of single- and double-U-tubes. For the circular-tube annulus, the correlation of Petuhkov and Roizen is used. For more information about the correlations we refer to Hellström [10].

The transient heat transfer from the internal wall of the pipes to the borehole wall is calculated using the thermal resistive-capacitive models (TRCM) derived by Bauer et al. [3]. These authors propose to extend the resistance model of Hellström for heat transfer in

the BHX (see [10]) to a dynamic model by adding capacities to it. For the case of a single U-tube, they also propose an empirical formula to approximate the multipole method of Bennet et al., using heat conduction shape coefficients and correction terms depending on the shank spacing divided by the borehole diameter. The correction terms are derived from an extensive set of simulations. The method is developed for coaxial, single U-tube and double U-tube types of borehole. The position of the capacities is calculated to be at the area center of the borehole with an equivalent single pipe.

Finally, the heat transfer from the borehole wall to the surrounding ground is calculated by discretizing the ground using a TRCM. The mesh is generated according to Eskilson's guidelines [9]:

$$\Delta r = [\Delta r_{min}, \Delta r_{min}, \Delta r_{min}, \beta \Delta r_{min}, \beta^2 \Delta r_{min}, \dots] ,$$

$$\Delta r_{min} = \min(\sqrt{\alpha \Delta t_{min}}, H/5) ,$$

with  $\alpha$  the diffusivity of the ground,  $H$  the depth of the borehole,  $\Delta t_{min}$  the minimum resolution time and  $\Delta r$  the size of the cell. The discretization has been tested with the analytical Cylindrical Source Model developed by Carslaw and Jaeger [7] and it shows very good agreement when the mesh is chosen fine enough. The width of the ground layer is by default equal to three meters but it can be adapted. The heat port at the external side of the layer is connected to a constant prescribed temperature equal to the initial undisturbed ground temperature. The heat flux at the external side of the ground layer is indeed very low from short simulation time.

## 3 Computation of the response function and aggregation method

The STM gives an accurate step response of the borefield as long as the diffusion length of the thermal process is small compared to the radius of its ground layer model or to the distance between the boreholes. The LTM is able to correctly compute the step response of the ground for a long time horizon as well as the interaction between the boreholes. It does not calculate, however, the borehole thermal resistance and its transient behaviour, contrary to the STM. The full response function is then obtained by lifting the LTM response to the STM response in the time interval where both models are still valid as shown in Fig. 2. As Javed mentions in his work [12], this interval is quite large (default value in model = 200 hours). Physically,

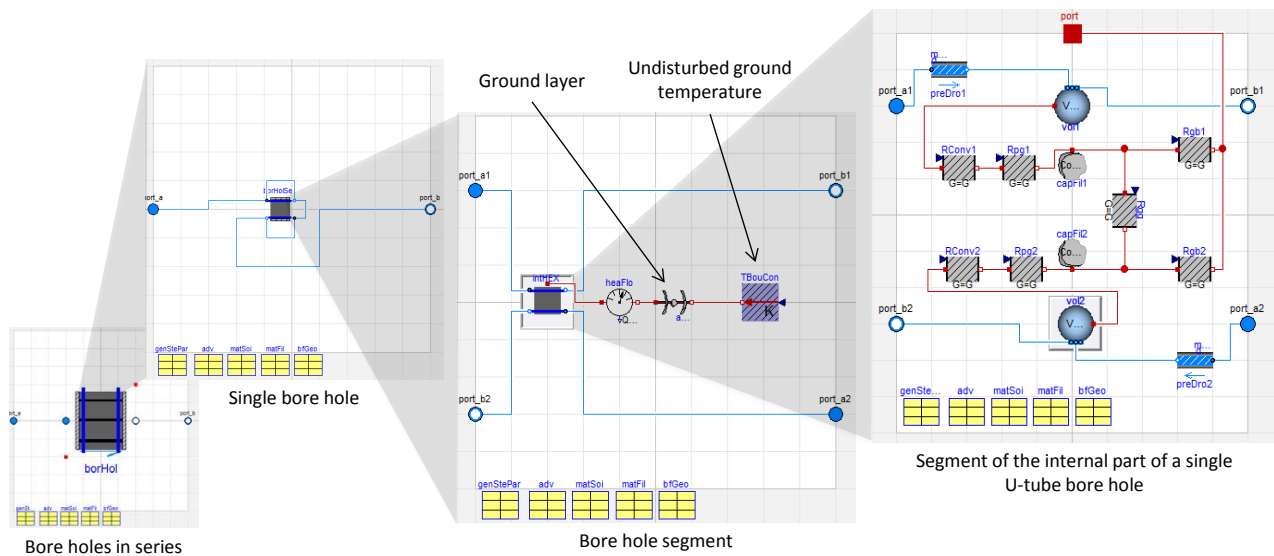


Figure 1: Implementation of the short-term model for boreholes in series in Modelica.

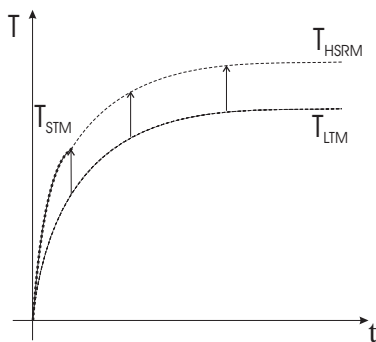


Figure 2: Combination of the long-term temperature step response ( $T_{LTM}$ ) with the short-term temperature step response ( $T_{STM}$ ) to compose the global temperature step response ( $T_{HSRM}$ ).

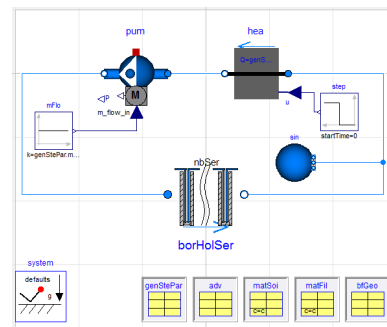


Figure 3: Model for the short-term temperature step response. The boreholes in series (*borHolSer*) are connected to a pump (*pum*) and to an ideal heater (*hea*). All the parameter values are stored in the records (bottom of the figure).

this interval begins when the transient behaviour of the BHX is over and it lasts until the interactions of the boreholes start to appear. The combination of both STM and LTM gives an accurate response function for both short- and long-term.

The response-function can be calculated at the start of each simulation or it can be priorly saved with a sample time equal to time resolution of the model. Until now, only the response of the STM is priorly saved in the implemented model in order to increase the computational speed but to avoid large files containing the full response function. The STM is connected to a pump and a prescribed heater/cooler from the Buildings library (see Fig. 3). A script-function automates the simulation of the STM and it writes the sampling values of its temperature response in the

Dat a package of the model as .txt file. The file is read at the initialization of the model in order to build the full response function of the HSRM.

As described above, g-functions and most of the analytical models give only a step response solution for the borefield. In order to model arbitrary input signals, the inputs need to be represented by a sum of time-shifted step signals and their responses should be superposed.

For minute-based multi-year simulations where the individual step response of each input step should be summed, this approach leads to enormous calculations. This problem is solved by using an aggregation method. The following paragraphs describe the technique of Claessons and Javed [12]. The notation has been adapted to gain clarity.

Assume that the discrete load input to the borefield is  $Q$  and the HCF temperature is  $T_f$ .  $Q$  and  $T_f$  can be written as:

$$Q_v^{(n)} := \begin{cases} Q[(n+1-v)h], & \text{if } v \leq n. \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

$$T_f(nh) - T_f(0) = \sum_{v=1}^{v_{\max}} \frac{Q_v^{(n)}}{Q_{\text{step}}} [T_{f,\text{step}}(vh) - T_{f,\text{step}}(vh-h)] \quad (6)$$

with  $v_{\max} \geq n$ ,  $h$  the discrete time-step,  $Q$  the discrete load and  $T_{f,\text{step}}$  the response function from HSRM with step load  $Q_{\text{step}}$ . Notice that the model assumes a uniform temperature at time 0.

The idea behind this aggregation is the following: the HCF temperature difference of the borehole system (from an initial steady state) at  $t = nh$  depends on the  $nh$  load pulses which have been applied to the borehole system from  $t = 0$  to  $nh$ . However, the influence of the pulses on the HCF temperature decreases the further they are from the observation time  $nh$ . If the pulses happened long before the observation time, the transient behaviour of the BHX has faded out, and only the net energy injection or extraction of the pulse is important. This net energy injection or extraction will indeed increase or decrease the global temperature of the borefield. An accurate profile of the load, far away from the observation time, is therefore not necessary. On the contrary, the load profile at times close to the observation time is important because they still influence the transient behaviour of the borefield and immediate surrounding ground.

Claesson and Javed proposed an aggregation algorithm grouping (i.e. taking the average of) the load pulses and their coefficients into cells of exponentially increasing size. The cells are themselves grouped into  $q$  levels. Each level has a given number of cells  $p_{\max}$  and each cell of a same level contains the same amount of load pulses  $R_q$ . Javed and Claesson propose to double the size of the cells at each level, in order to have the same number of cells in each level and finally in order to choose this number of cells per level according to the desired accuracy (a higher number of cells per level gives a more detailed load profile but penalizes the computational efficiency).

Eq. 6 is now rewritten to implement the aggregation method. Notice that the temperature difference of the HCF between two time steps in Eq. 6 divided by the amplitude of the step load  $Q_{\text{step}}$  can be considered

as the transient thermal resistance of the borehole for that particular time. Let us define the transient thermal resistance  $R_v$  and the dimensionless factor  $\kappa_v$  as:

$$R_v = \frac{T_{f,\text{step}}(vh) - T_{f,\text{step}}(vh-h)}{Q_{\text{step}}} \quad (7)$$

$$\kappa_v = \frac{T_{f,\text{step}}(vh) - T_{f,\text{step}}(vh-h)}{T_{f,\text{step}}(\infty)} = \frac{R_v}{R_{ss}}. \quad (8)$$

Eq. 6 can now be rewritten as:

$$T_f(nh) - T_f(0) = R_{ss} \sum_{v=1}^{v_{\max}} Q_v^{(n)} \kappa_v. \quad (9)$$

with  $R_{ss}$  the steady state thermal resistance.

As explained above, the aggregation is consisting of  $q_{\max}$  levels, each composed of  $p_{\max}$  cells which have a level-dependent width  $R_q$  defined as:

$$R_q := 2^{q-1} \quad \text{for } q = 1, \dots, q_{\max}. \quad (10)$$

The number of pulses covered by the aggregation is then:

$$v_{\max} := \sum_{q=1}^{q_{\max}} R_q p_{\max} \geq n_{\max}. \quad (11)$$

Define  $v_{q,p}$  as the number of pulses covered from cell 1 at level 1 till (including) cell  $p$  at level  $q$ :

$$v_{q,p} := p R_q + \sum_{i=1}^{q-1} R_i p_{\max}. \quad (12)$$

Define the function  $v(q, p, r)$  numbering each pulse, starting from pulse 1 in cell 1 at level 1:

$$v(q, p, r) := v_{q,p} - R_q + r \quad \text{for } q = 1, \dots, q_{\max}, \\ p = 1, \dots, p_{\max}, \\ r = 1, \dots, R_q.$$

These different definitions are illustrated in Fig.4.

Using these definitions, Eq. 9 can be rewritten as:

$$T_f(nh) - T_f(0) = R_{ss} \sum_{q=1}^{q_{\max}} \sum_{p=1}^{p_{\max}} \sum_{r=1}^{R_q} Q_{v(q,p,r)}^{(n)} \kappa_{v(q,p,r)} \quad (13)$$

Now we apply the aggregation technique by approximating the last sum of Eq. 13 by

$$\sum_{r=1}^{R_q} Q_{v(q,p,r)}^{(n)} \kappa_{v(q,p,r)} \approx \left[ \frac{\sum_{r=1}^{R_q} Q_v^{(n)}}{R_q} \right] \sum_{r=1}^{R_q} \kappa_{v(q,p,r)} := \bar{Q}_{v(q,p)}^{(n)} \bar{\kappa}_{v(q,p)} \quad (14)$$

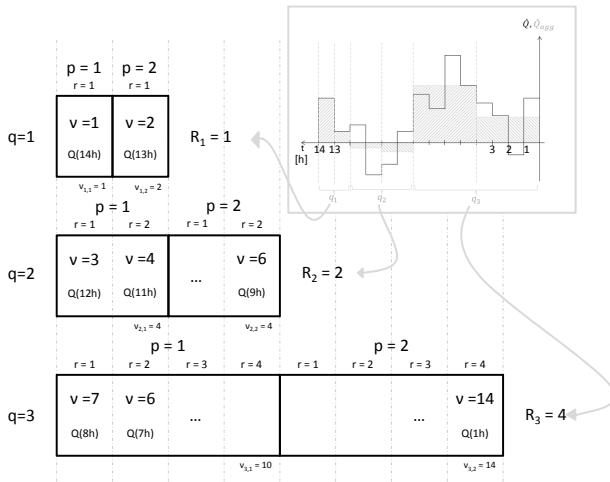


Figure 4: Illustration of the aggregation method for a load of 14 hours with time steps (pulses) of one hour. The number of levels is three and each level has two cells. The size of the cells is doubled at each level.

Finally the aggregation of Eq. 9 gives:

$$T_f(nh) - T_f(0) \approx R_{ss} \sum_{q=1}^{q_{max}} \sum_{p=1}^{p_{max}} \bar{Q}_{v(q,p)}^{(n)} \bar{\kappa}_{v(q,p)}. \quad (15)$$

The term  $\bar{\kappa}_{v(q,p)}$  is a matrix with the transient thermal resistance of each cell of the aggregation and it is independent of the load. This matrix is currently calculated at the initialization of each simulation. For repetitive simulations using the model and the same simulation length, the matrix can be priorly calculated and saved to gain significant calculation time. The term  $\bar{Q}_{v(q,p)}^{(n)}$  is a vector with a length equal to the number of aggregation cells and which is composed of the aggregated past load pulses. At each new discrete simulation time, a new load pulse needs to be added and the previous pulses need to be shifted in the  $\bar{Q}_{v(q,p)}^{(n)}$  vector. This means re-calculating the whole vector. Claesson and Javed developed a method which avoids this time consuming re-calculation by updating instead the load vector from the previous time step. The method is based on the shift of each cell to the next one and it has been applied to our model. An error, however, is introduced due to mixing in the cells. Claesson and Javed concluded after a detailed study that the error can be neglected. For example, in case of a simulation of 20 years using the aggregation method with each level having 5 cells, the error compared to the non-aggregated solution is lower than 0.1 K (for more information about the method and accuracy, see Claesson and Javed [8]).

Note that the left-hand term of Eq. 15 is only an

approximation of its right-hand term due to the approximation made in Eq. 14. The error, however, is negligible if the number of cells is sufficiently high. By default, the number of cells by level is five and the size of the levels increase exponentially with base two.

## 4 Model validation

The STM and the LTM have been verified by their respective developers. To avoid coding error and to check and generalize the validity of the model, the model verification has been extended.

The STM is compared to the widely used sandbox experiment of Beier et al. [5]. These authors have carefully performed a thermal response test using a U-tube BHX. The U-tube is grouted into an aluminium pipe of 18 meters long which is placed into a box filled with homogeneous sand. An electrical heater injects a constant heat rate to the HCF and a pump insures a constant flow rate. All ground and grout properties are presented in the paper, except the heat capacities. The ground capacity has been estimated by Beier using a best fit method ( $c_v = 3.2MJ/m^3K$ ) [4]. For the grout a heat capacity of  $4MJ/m^3K$  is used. The HCF temperature is measured at the in- and outlet as well as the BHX wall and sand temperatures at various depths. It should be noted that the aluminium pipe around the grout acts as a thermal fin which reduces the borehole thermal resistance by evening out its wall temperature. The HCF temperature should therefore be lower for the experiment than for the models which do not take this fin effect into account (see Lamarche 2010 [13]). Fig. 5 compares the average of the in and outlet temperatures of the HCF for the case of the experiment, the Buildings model, TRNSYS model (type 557a, DST) and the implemented HSRM. The Buildings model dynamics is clearly too slow. This is due to the position of the lumped capacity of the grout, as illustrated by Bauer et al [3]. In the Buildings model, the grout capacities are positioned at the pipe wall instead of the area center of each grout zone. Adapting the capacity location (which requires also the adaptation of the resistances), the problem is solved (*Buildings adapted*). TRNSYS DST model and HSRM give similar results. DST, however, does not incorporate the short-term thermal dynamics of the fluid, contrary to the new model HSRM.

The LTM is verified using the well known g-function developed by Eskilson and the infinite cylindrical heat source (CHS) solution for different configurations (the data are taken from the paper of Bertag-

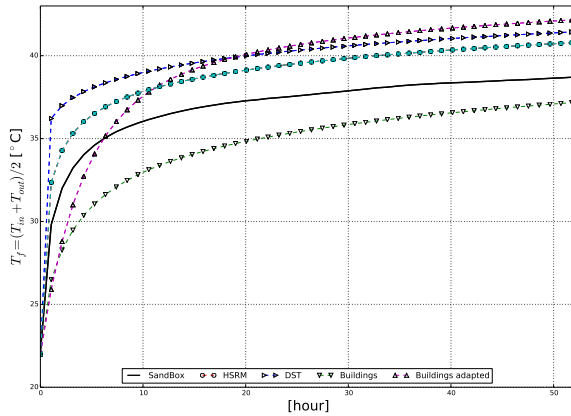


Figure 5: Comparison between the heat carrier fluid temperature from the sandbox experiment ([5]), the borehole model from the Buildings library and its adapted version, type 557a of TRNSYS (DST) and the new hybrid model (HSRM).

nolio [6]). Fig. 6 illustrates the case of a 110 meter deep single borehole. The error of the implemented model compared to the  $g$ -function never exceeds 0.11 K during the 25 year-long simulation. The difference is caused by the so-called end effect of the borehole because the analytical solution uses a finite line-source approximation whereas the Eskilson finite volume model is three-dimensional (boundary difference at the foot of each borehole). The CHS model is clearly unable to model the end effect. Fig. 7 illustrates the case of a borefield with a square 8x8 configuration, respectively. The length of the boreholes is 110 meters and the relative distance between the boreholes to length ratio equals 0.05. Due to the very compact configuration, a large error appears, as Malayappan and Spitler warned for [14]. The error comes from the assumption that each borehole injects or extracts the same amount of heat, regardless of its relative position in the borefield. In reality, the boreholes at the edge of the borefield will inject/extract more than the center ones and as a consequence, the average borefield temperature will be lower. The end effect error is negligible compared to the large error ( $> 7K$  after 25 years for this case) introduced by the homogeneous heat source approximation. However, if the borefield is dissipative enough, the model shows very good results (e.g. see Fig. 8 for a line configuration of eight boreholes). For simulation with yearly thermal ground balance (amount of injected heat = amount of extracted heat), the configuration error is partly counteracted and it will not cause significant accuracy issues.

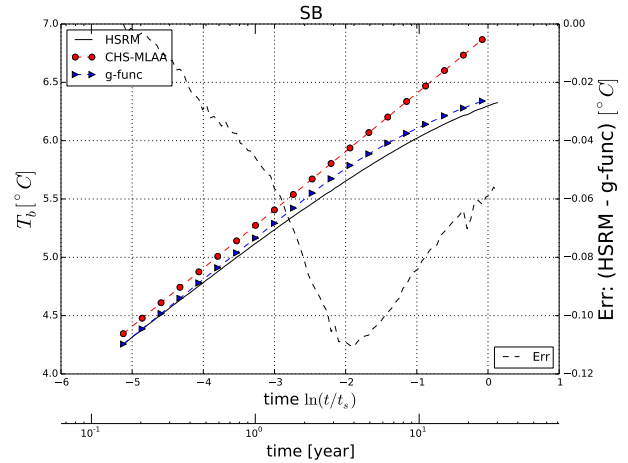


Figure 6: Average temperature step response of the borehole wall of a single borehole calculated by the  $g$ -function ( $g$ -func), the infinite cylindrical source with aggregation method (CHS-MLAA) and the new hybrid model (HSRM).

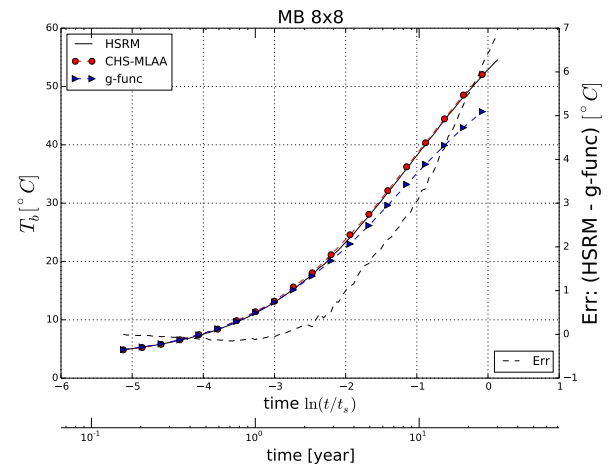


Figure 7: Average temperature step response of the borehole walls of a borefield with 64 boreholes in filled square configuration ( $B/H=0.05$ ) calculated by the  $g$ -function ( $g$ -func), the infinite cylindrical source with aggregation method (CHS-MLAA) and the new hybrid model (HSRM).

## 5 Example

This section describes an example of a borefield subjected to a varying non-symmetric load with a time-step of 4 hours proposed by Bernier et al [6]. The CPU and the fluid temperature of the Buildings model and those of the HSRM model are compared for a simulation of one year in the case of a single borehole and the case of three boreholes in series (Fig. 11). The Build-

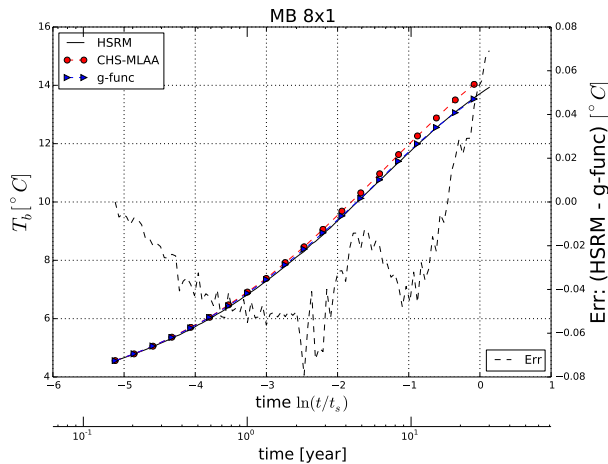


Figure 8: Average temperature step response of the borehole walls of a borefield with 8 boreholes in line configuration ( $B/H=0.05$ ) calculated by the  $g$ -function ( $g$ -func), the infinite cylindrical source with aggregation method ( $CHS$ - $MLAA$ ) and the new hybrid model ( $HSRM$ ).

ings model is composed of the Buildings component `Boreholes.UTube`, an ideal heater and a pump (see Fig. 9 for the case of three boreholes in series). The  $HSRM$  model uses the same setup but the Building boreholes are replaced by the  $HSRM$  (Fig. 10). A step response of 200 hours is calculated with the STM prior to the simulation, in order to calculate the short term part of the response function. The interaction between the boreholes is taken into account by the  $HSRM$  but not by the Buildings model.

As seen above, the Buildings model underestimates the borehole resistance which is also visible in Fig. 11 where the fluctuations of HCF temperature of the Buildings model have a smaller amplitude than those of the  $HSRM$  model.

The analysis of the CPU times illustrates very clearly the difference between the models. In the case of a single borehole, the  $HSRM$  model is about twelve times faster than the Buildings model. The  $HSRM$  has a longer initialization time due to the calculation of the aggregation matrix, but it calculates the temperature response very fast. In the case of three boreholes in series, the  $HSRM$  is about 60 times faster. The initialization time is longer than for a single borehole because the superposition of the temperature field of the boreholes needs to be calculated. However, once the aggregation matrix is calculated, the calculation time is the same for any configuration. This is not the case for the Buildings model.

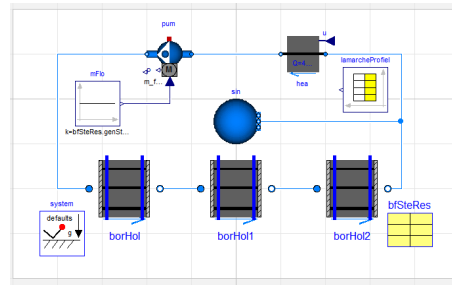


Figure 9: Model of three boreholes in series with a variable heat load (described by Bernier et al. [6]) and a constant mass flow rate, using components of the Buildings library.

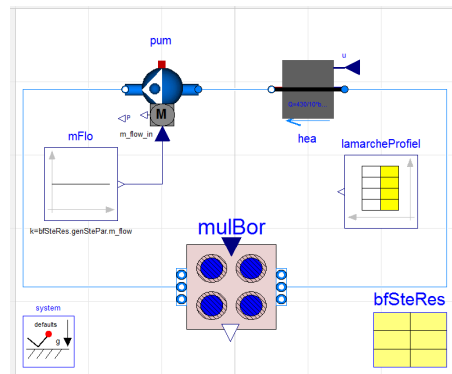


Figure 10: Model of three boreholes in series with a variable heat load (described by Bernier et al. [6]) and a constant mass flow rate, using the new borefield ( $multBor$ ) model and components of the Buildings library.

## 6 Conclusion

A new hybrid model for borefields with arbitrary configuration having both short-term (minutes) and long-term accuracy (decades) has been successfully developed and implemented in Modelica. The model has been validated for both short- and long-term. Thanks to its aggregation method, the implemented model is about twelve times faster than the borehole model of the Buildings library for the case of a single borehole and about 60 times faster for the case of three boreholes in series. The long-term accuracy of the model decreases for compact borefield configuration. This can be solved by plugging a  $g$ -function in the model instead of calculating the temperature step response.

## 7 Acknowledgment

The authors acknowledge the financial support by IWT and WTCB in the name of the IWT-VIS Traject



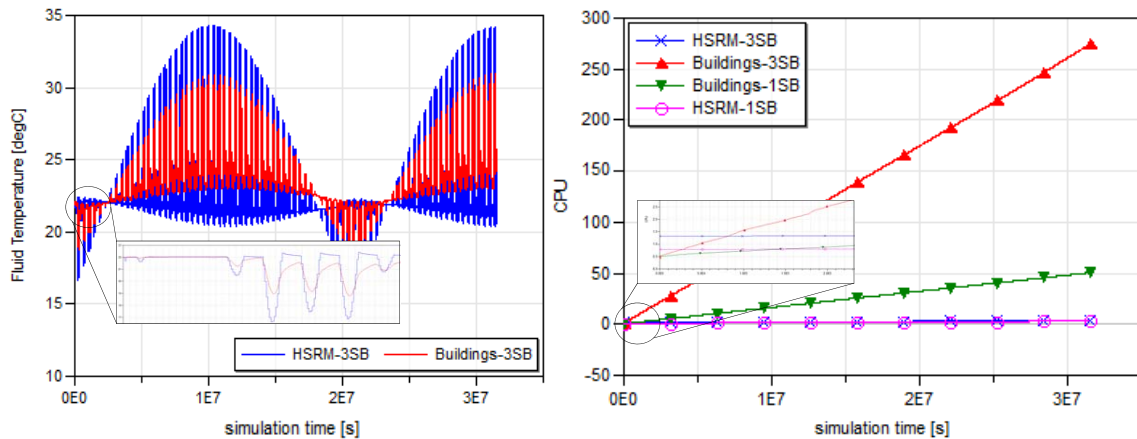


Figure 11: Left: CPU comparison between the new model (*HSRM*) and the model from the Buildings library (*Buildings*) for a single borehole (*1SB*) and for three boreholes in serie (*3BH*). Right: heat carrier temperature for *HSRM-3BH* and *Buildings-3BH*.

SMART GEOTHERM focusing on integration of thermal energy storage and thermal inertia in geothermal concepts for smart heating and cooling of (medium) large buildings. Moreover this study is part of the development work performed within IEA-ECB-Annex 60 on new generation computational tools for buildings and community energy systems based on the Modelica and Functional Mockup Interface standards.

## References

- [1] *EnergyPlus, Getting started with EnergyPlus*.
- [2] *TRNSYS 17, a transient system simulation program*.
- [3] D. Bauer, W. Heidemann, H. Müller-Steinhagen, and H.-J. G. Diersch. Thermal resistance and capacity models for borehole heat exchangers. *internal journal of energy research*, 35:312–320, 2010.
- [4] R. A. Beier. Transient heat transfer in a u-tube borehole heat exchanger. paper accepted for publication in *Applied Thermal Engineering*, 19-09-2013.
- [5] R. A. Beier, M. D. Smith, and J. D. Spilter. Reference data sets for vertical borehole ground heat exchanger models and thermal response test analysis. *Geothermics*, 40:79–85, 2011.
- [6] S. Bertagnolio, M. Bernier, and M. Kummert. Comparing vertical ground heat exchanger models. *Building Performance Simulation*, 1:1–15, 2012.
- [7] H. S. Carslaw and J. C. Jaeger. *Conduction of Heat in Solids*. Oxford University Press, UK, 1959.
- [8] J. Claesson and S. Javed. A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. *ASHRAE Transactions*, 118, Part 1, 2012.
- [9] P. Eskilson. *Thermal analysis of heat extraction boreholes*. PhD thesis, Dep. of Mathematical Physics, University of Lund, Sweden, 1987.
- [10] G. Hellström. *Ground heat storage: thermal analyses of duct storage systems (Theory)*. Dep. of Mathematical Physics, University of Lund, Sweden, 1991.
- [11] K. Huchtemann and D. Müller. Advanced simulation methods for heat pump systems. *Proceedings 7th Modelica Conference, Como, Italy*, 2009.
- [12] S. Javed. *Thermal modelling and evaluation of borehole heat transfer*. PhD thesis, Dep. Energy and Environment, Chalmers University of Technology, Göteborg, Sweden, 2012.
- [13] L. Lamarche, S. Kajl, and B. Beauchamp. A review of methods to evaluate borehole thermal resistance in geothermal heat-pump systems. *Geothermics*, 39:187–200, 2010.

- [14] V. Malayappan and J.D. Spitler. Limitations of using uniform heat flux assumptions in sizing vertical borehole heat exchanger fields. In *Proceedings of Clima, June 16-19. Prague, 2013*.
- [15] Haruhiko Okumura. *New Algorithm handbook in C language*. Gijyutsu hyouron sha, Tokyo, p227, 1991.
- [16] T. Schmidt and G. Hellström. Superposition borehole model, working paper on usable tools and methods. Technical report, Nordon, nordic energy research, February 2005.
- [17] M. Wetter and A. Huber. Trnsys type451, vertical borehole heat exchanger, ews model, version 3.1, model description and implementating into trnsys. Technical report, TranssolarGmbH, Stuttgart , Germany., 1997.
- [18] M. Wetter, W. Zuo, T. Nouidui, and X. Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 0:1–18, 2013.

# Superheat Control with a Dynamic Inverse Model

Andreas Varchmin<sup>1</sup> Manuel Gräber<sup>2</sup> Wilhelm Tegethoff<sup>1,2</sup> Jürgen Köhler<sup>1</sup>

<sup>1</sup> Technische Universität Braunschweig, Institut für Thermodynamik

<sup>2</sup> TLK-Thermo GmbH

## Abstract

Superheat control has influence on the coefficient of performance (COP), the stability and the compressor endurance of a vapor compression cycle. In an increasing number of applications electronic expansion valves are used. This leads to more complex control tasks. It raises the question if simulation models can be used for feedforward control to fulfill this function. For building a feedforward control structure a simulation model needs to be inverted. In this paper a continuous submodel of a refrigeration cycle, consisting of models for expansion valve and evaporator, is inverted. The resulting controller is tested in a model-in-the-loop environment and applied on an automotive refrigeration cycle. The advantage of a dynamic inverse model in contrast to a static one is pointed out. Also the results are compared to a standard PI controller.

*Keywords: inverse models; superheat control; vapor compression cycles; feedforward control*

## 1 Introduction

Superheat has several effects on refrigeration and heat pump cycles. The coefficient of performance (COP), the stability and the compressor endurance are dependent from a reasonable degree of superheat. Superheat control is a challenging task since the system dynamics of a refrigeration cycle are highly nonlinear and the control targets can be contrary. A higher degree of superheat leads to a smaller COP but is more secure against damage from liquid drops inside the compressor. Moreover due to the inherent characteristics of vapor compression cycles there exists an instability region at small degrees of superheat that expresses in a fluctuating temperature. This region can be described by the minimum stable superheat line (MSS) [1, 2]. Hence, the target of control structures is to set a small degree of superheat with considering the robustness against crossing the MSS line.

In the last years electronic expansion valves (EXV) are used in an increasing number of applications, e.g. electronic vehicles. With EXVs superheat and thus stability and COP can be controlled directly in contrast to thermostatic expansion valves (TXV) or orifice valves. New control approaches can benefit from this additional degree of freedom.

In the last decade a lot of effort has been made to build models for describing the behavior of vapor compression cycles. Model libraries like *TIL* or *Air Conditioning* have been developed to simulate various systems. It seems consequential to use this knowledge not only for simulation but also for control. One approach for model based control is inverting the simulation models.

Using inverse models for feedforward control is common in control theory [3]. Inverting a model means to swap inputs and outputs and reform the model equations so that the inputs can be calculated when having knowledge of the outputs. In this way the actuating variables can be calculated depending of the desired system output. Modelica is convenient for inverting models since it is an equation-based language instead of signal-oriented. One of the main purposes of every modelica tool is to transform ODE- and DAE-systems. This is exactly the challenge of inverting models. For a convenient introduction in inverting models in Modelica see [4].

Often static models are used for feedforward control. In this paper a dynamic model is proposed for controlling the superheat. This has advantages if the desired system output is not a constant or if measurable dynamic disturbances act on the system.

## Structure of the Paper

In section 2 a continuous system model of the refrigeration cycle is introduced. One part of the model, a submodel consisting of expansion valve and evaporator, is needed for feedforward control. The complete system model is used for model-in-the-loop tests and as an observer during measurement. In section 3 the

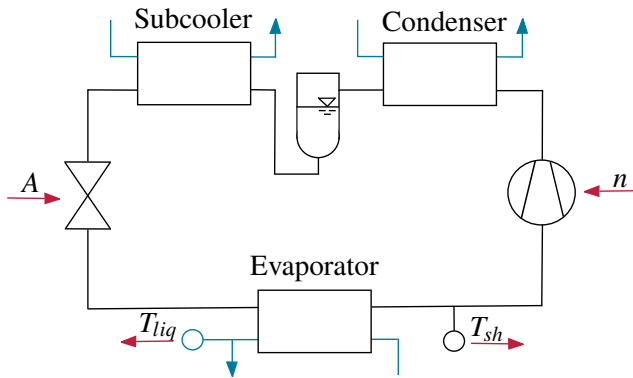


Figure 1: Modelled Refrigeration Cycle

inversion of the submodel is shown. The developed inverse model is tested in a model-in-the-loop environment in section 4. The potential of the transient inverse model in relation to a static one is also presented. Section 5 describes the test bench setup and shows how the complete system model is used as an observer. Measurement results of the inverse feedforward control are shown in section 6 as well as closed loop control where the inverse model is coupled with a PI controller. The measurement results are compared with regular PI control results.

## 2 System Model of the Refrigeration Cycle

The modelled system is a basic refrigeration cycle consisting of a compressor, an EXV, three plate heat exchangers and a receiver (see figure 1). R134a is used as refrigerant. The compressor is a variable speed scroll compressor. On the high pressure level there are two heat exchangers: condenser and subcooler. In between lies the receiver to ensure that subcooling exists in all operating points. In the outlet of the evaporator the degree of superheat is measured. A 50/50 mixture of water and glycol flows in the secondary paths of all heat exchangers. In terms of control the valve opening  $A$  and the compressor speed  $n$  can be defined as inputs and the actual degree superheat  $T_{sh}$  and the temperature of the cooled liquid  $T_{liq}$  as outputs.

Most of the model approaches are based on the model library *TIL* (see [5]). For the inversion new models with functions and interfaces were developed. All physical properties are calculated with *TILMedia* by use of refrigerant and liquid objects or explicit function calls (see [6]).

The dynamic compressor model is loss-based and has a suction and a discharge volume where the phys-

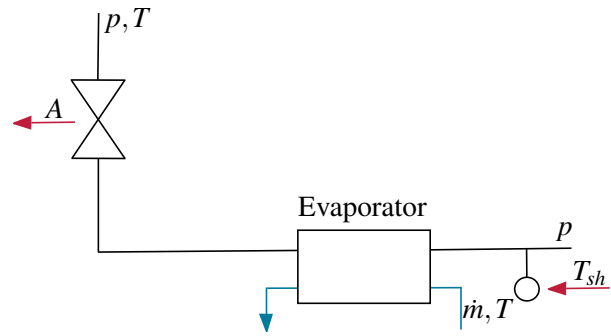


Figure 2: Inverse Model

ical properties of the refrigerant are calculated. The model parameters are fitted to the compressor in the test bench described in section 5. The EXV model originates from Bernoulli's principle and is therefore a static model. Delays that may come from the actuator of the valve are not considered even if they can not be neglected. Section 3 deals with this problem. All heat exchangers are modelled with a variable number of finite volumes. Per volume exist three cells, one for the refrigerant, one for the liquid and one for the wall dividing the fluid paths. The refrigerant cell includes transient mass, energy and momentum balances. The time derivative of pressure is constant over all refrigerant cells of a pressure level [7]. The liquid cell includes transient balances for mass and energy. Pressure drop is neglected in both liquid and refrigerant cells. The wall cell includes a transient energy balance. The receiver between the two high pressure heat exchangers is modelled as a volume with transient mass and energy balances and a varying filling level.

## 3 Inverting the Submodel

For representing the complete system dynamics for feedforward control it is desirable to invert the whole system model described in the previous section. With linear analysis of the refrigeration cycle it can be shown that an inversion of the complete model is not reasonable due to possible non-minimum phase characteristics. This would lead to an unstable system model. Hence, in this paper only a submodel is used for feedforward control. The inverted model consists of the evaporator and valve models.

The resulting inverse model is shown in figure 2. The easiest way creating it is using the MSL Block *InverseBlockConstraints* that changes in- and outputs. The desired superheat  $T_{sh}$  is now an input and the resulting valve opening  $A$  an output. The outlet liquid temperature of the evaporator, that is equivalent to

the cooling capacity, shall be controlled by the compressor speed and is therefore not needed for superheat control. Since the inverse model does not include the whole system dynamics boundary conditions are needed. One combination of possible boundaries is shown in figure 2. High and low pressure as well as the refrigerant inlet enthalpy of the valve are needed to simulate the model. Moreover the mass flow rate and temperature of the evaporator inlet need to be known. This information can be provided by simulation data when using the inverse model in a model-in-the-loop environment. For use as a hardware controller measurement data and/or data from an observer can be used (see section 5).

The valve opening is dependent from the opening actuator time delay. This delay can be approximated as a first-order time delay. Reforming the ODE system gets more complicated because of an additional continuous state that makes it necessary to derive more equations and to give access to second derivatives with respect to time of physical properties. For these reasons including the valve actuator delay was not possible at the time of writing this paper.

The inverse model forms an ODE with 15 continuous states. For calculating the saturation temperature dependent from pressure, that is needed for calculating the superheat, a function from TILMedia is used. Also the time derivative of this function is provided. For comparison also a (nearly) stationary inverse model is build. When eliminating the continuous states several systems of nonlinear of equations result. To prevent these hardly solvable systems the capacities of the continuous states are chosen very small so that a nearly stationary model is generated.

## 4 Simulation Results

This section starts with results regarding the sole inverse model and continues with model-in-the-loop (MiL) tests. In figure 3 the response of the inverse model to a change of the superheat is shown. The superheat follows the reference trajectory besides a small difference. This comes from a filter that smoothes the reference trajectory to make it differentiable. Otherwise the model would not be invertible (see also [4]). The actuating variable, the valve opening area, fluctuates mainly at the changes of the reference trajectory derivative at 0s and 5s. A real-world EXV may not be able to follow the actuator trajectory. The high peaks, especially at 5s and 6s, are mostly important to follow the gradient of the set point change.

In most applications it is not necessary to follow this gradient exactly.

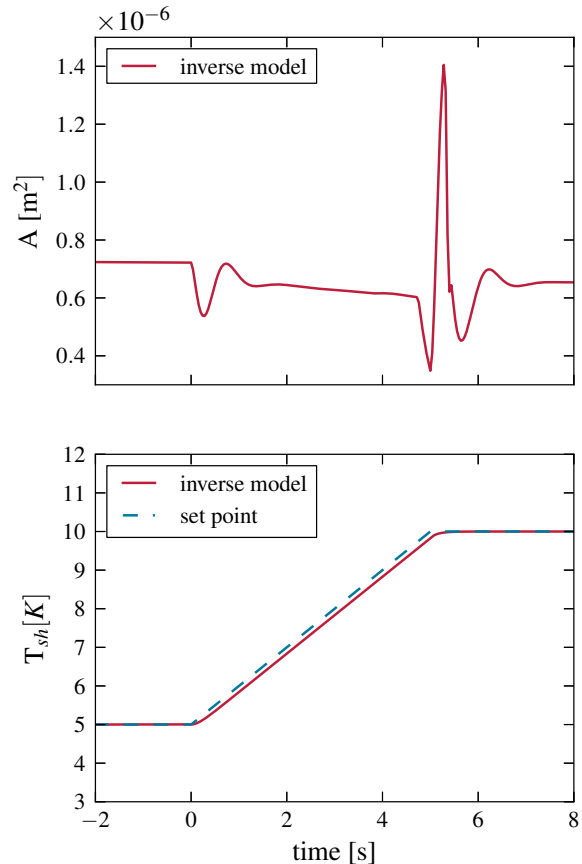


Figure 3: Behavior of Inverse Model to Setpoint Change of Superheating Degree

Figure 4 shows a set point change in a MiL environment. A transient and a static inverse model work as feedforward controllers in this example. It is obvious that the transient model can make the system model follow the reference trajectory a lot better. The static model is not able to represent the dynamics of the system and can not follow the reference trajectory. In both simulations the set point is not reached exactly. This arises from model uncertainties between inverse and system models. The valve opening fluctuates more than in the previous simulation of the sole transient model. Because of the two pressures and the valve inlet enthalpy there exists a coupling between the whole system model and the inverse model in MiL simulations. This leads to a more dynamic system. Summarizing the transient feedforward control offers a big potential for the response to setpoint changes.

Besides the response to setpoint changes the behavior towards disturbances is a main attribute for reliable and robust control. One of the advantages of feedfor-

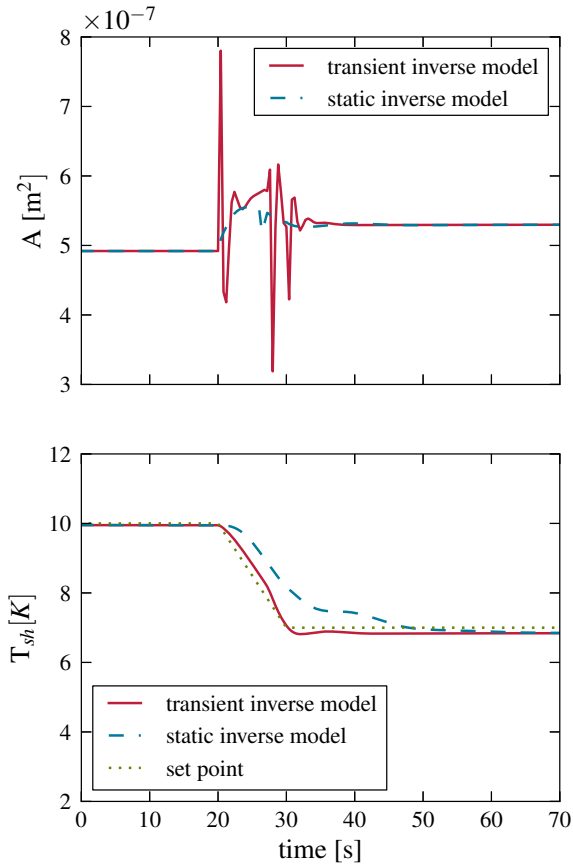


Figure 4: Comparison between Static and Transient Inverse Model Control (Model-in-the-Loop)

ward control combined with feedback control is that these behaviors can be divided. Mostly the feedforward part is mainly responsible for responses to set-point changes so that the feedback part can be designed for reacting on disturbances and model uncertainties. If the disturbances are measurable also feedforward control can handle them and even improve the control scheme. In the described process model disturbances can be imagined as changes in input temperatures of the secondary loop cycles or as changes in the compressor speed. The compressor speed can be controlled as well and is therefore known. A change in compressor speed has influence on pressures and mass flow rate of the refrigerant cycle. This information is provided to the inverse model that can react on this disturbance. An example is shown in figure 5 where the compressor speed is raised from 2000 rpm to 3000 rpm at time 0. Results are shown for feedforward control including a model uncertainty and a PI controller. The PI controller reacts as recently as the measurement signal deviates from the set point. The superheat rises from 10 K to approximately 15 K. The feedforward

controller reacts at time 0 when the compressor speed rises. The superheat stays under 11 K. Even if the PI controller could be optimized further for reacting on this disturbance it can never react as fast and precise as the inverse model.

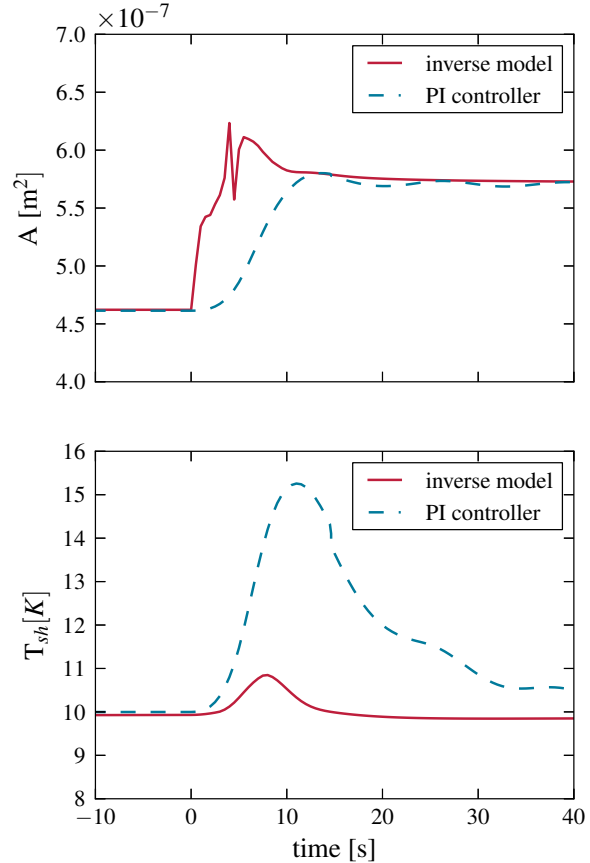


Figure 5: Behavior of the System Model towards Disturbance (Change of Compressor Speed): Comparison between Inverse Model Control and PI Controller

## 5 Test Bench Setup

The developed controller is tested on a refrigeration cycle that has the dimension of an automotive air conditioning application (see figure 6). The compressor is a direct current scroll compressor with variable speed as it is used in electric vehicles. As heat source and sink controllable secondary loop cycles filled with a glycol water mixture are connected with the plate heat exchangers. The EXV is driven by a stepper motor with constant adjusting speed. The motor needs a voltage signal for positioning the actuator. The voltage is calculated by a linear correlation with the valve opening area as input. The degree of superheated is measured with two thermocouples, one at the inlet and

one at the outlet of the evaporator. The difference of the measured temperatures will equal the superheat if there exists two-phase refrigerant at the inlet which is the case during all measurements. The measured inlet temperature can be verified by the measured inlet pressure that is bound to the evaporation temperature.

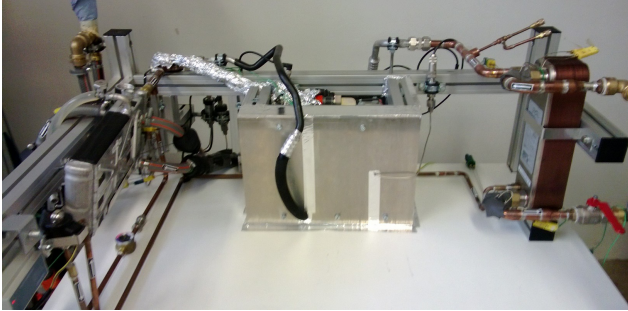


Figure 6: Test Bench with compressor, evaporator, condenser and expansion valve

A system model of the complete cycle (see section 2) is used as an observer. The current compressor speed and measured liquid temperatures and mass flow rates at evaporator, condenser and subcooler are made available to the observer. With this information it calculates high and low pressure and the valve inlet enthalpy that are needed by the inverse submodel.

The controller, consisting of the described inverse model for feedforward control and an optional PI controller for closing the control loop, and the observer are modelled in Dymola. The control scheme is shown in figure 7. Dymola is coupled with Labview in a co-simulation environment. Labview reads the measurement data and sends process inputs, e.g. valve opening area. One time step has a length of 0.1 s.

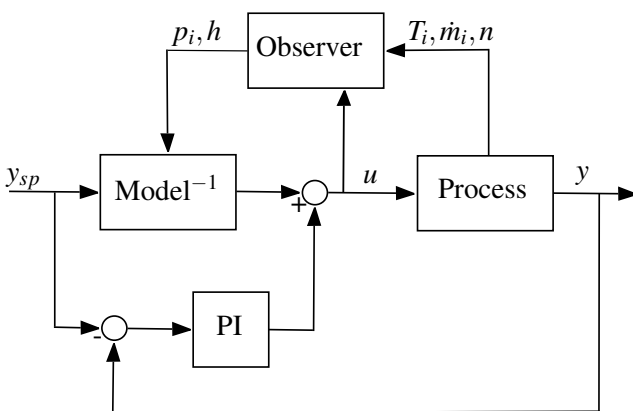


Figure 7: Inverse Model in Combination with PI Controller for Closed Loop Control

## 6 Measurement Results

In this section the developed model-based controller is applied to the test bench described previously. Results with the lone inverse model and combined with an PI controller are shown. For comparison also results generated with a lone PI controller are shown.

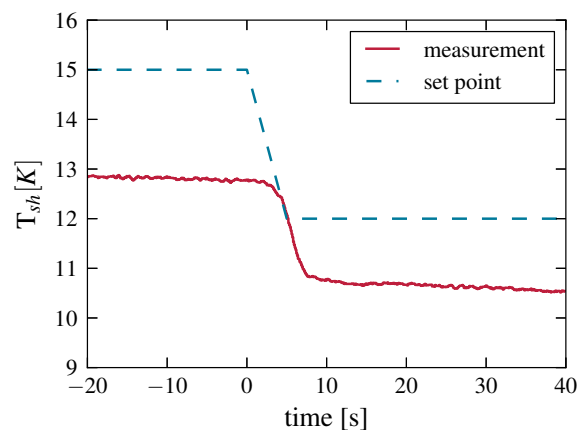
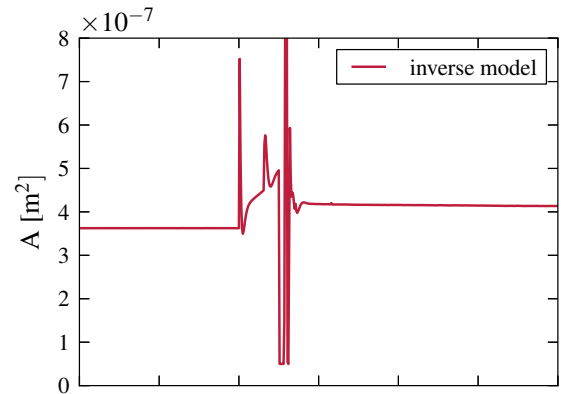


Figure 8: Comparison of model-based controller to setpoint change

In figure 8 the refrigeration cycle behavior to a set point change is shown. The setpoint follows a five second lasting ramp from 15 K superheat to 12 K. At 0 s the degree of superheated lies at approximately 13 K. The deviation arises from model uncertainties. In particular the uncertainties come mainly from a differing mass flow and unexact calculation of the heat transfer coefficients. Since no feedback loop is implemented in this case there is a stationary deviation. The sensor value drops nearly as fast as the setpoint and reaches its stationary value after circa six seconds. The measured superheat change is smaller than the demanded which also results from the described model uncertainties. The shown valve opening area does not picture the achieved opening area but the demand of the in-

verse model. The high peaks arise from the harsh bend in the set point ramp and the missing actuator delay in the inverse model. These peaks are filtered before directing the signal to the valve. Omitting the peaks the valve opening area rises nearly linear, then drops to a point beneath the new stationary value and rises again to the new stationary value. Summarizing apart from an error in stationary values the inverse model feedforward controller can change the degree of superheat fast and as demanded.

In the following the feedforward controller is combined with a PI controller to compensate the mentioned model uncertainties. There exist several rules to design the parameters of a PI controller when combined with feedforward control. In this case the feedforward controller gets information from the observer, especially the actual pressures. This means that the inverse model, even if called feedforward controller in this paper, has a kind of feedback loop implemented. High and low pressure react on a changing valve opening which for its part has influence on pressures. Hence, designing the PI controller is more challenging and standard design rules can not be used without adaptation. In this paper the parameters of the PI controller for combination with the inverse model are therefore iteratively chosen and will not be optimal.

For comparison with a standard PI controller typical design rules can be used for parametrization. In this paper the from Åström and Hägglund as a Ziegler-Nichols replacement introduced *AMIGO* method [8] is applied. For using this approach a measured step response is needed. The aim is to maximize the integral gain by modeling the step response with a dead time and a first-order block. Hence, three parameters have to be tuned to represent the step response. With knowledge of these parameters the PI control parameters can be obtained by simple, explicit equations.

In figure 9 a comparison between the developed model-based controller and a lone PI controller is shown. Compared to the lone feedforward control (fig. 9) the measured superheat meets the set point since the integral gain can compensate model uncertainties. The developed controller is able to drop the superheat more quickly but does undershoot the new setpoint a bit. This probably comes from the non-optimal cooperation of inverse model and PI controller. Improving this could lead to a fast and sharp response as with the lone model-based controller. The PI controller is slower but does less undershooting. It is important to mention that the PI controller is tuned for this exact operating

point. Providing this quality for other operating points would mean to have an at least two-dimensional gain scheduling to compensate different compressor speeds and evaporator inlet liquid temperatures. The actuating variable in figure 9 is shown as voltage since the lone PI controller does not give an opening area value. It can be seen that the model-based controller can calculate a more transient actuator signal.

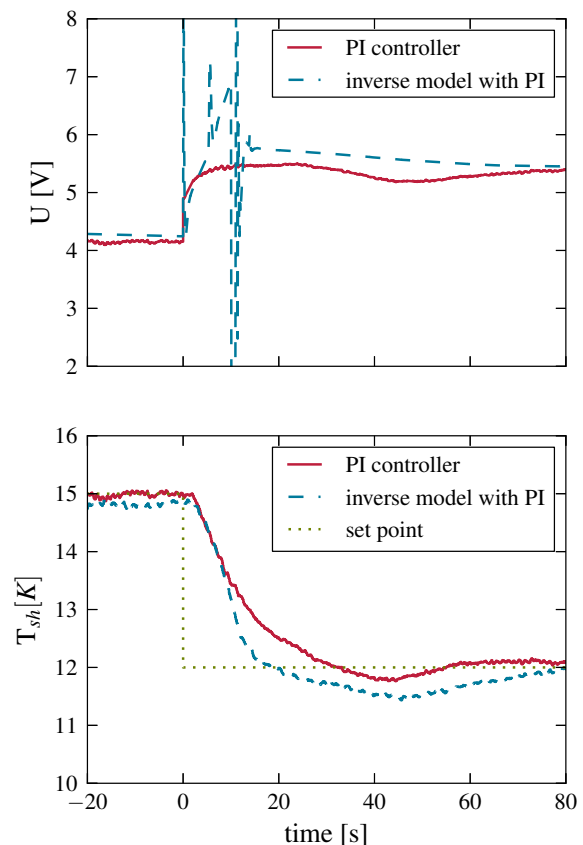


Figure 9: Comparison between reactions of model-based controller (including PI controller) and PI controller to setpoint change

## 7 Conclusion

In this paper a superheat controller based on an inverse model of valve and evaporator was developed and tested in simulations and measurements. The controller can react faster on set point changes than an optimized PI controller. The biggest advantage is the compensating reaction on measurable disturbances like changing compressor speed / cooling capacity. In contrast to PI control the model-based controller works in several operating points without parameter changes or gain scheduling. Further improvements



can be accomplished by including the valve actuator delay in the inverse model. Furthermore the interaction with a PI controller for compensating on model uncertainties and non-measurable disturbances can be enhanced.

## 8 Acknowledgements

This work has been supported by the German Ministry BMBF in the *Reflex Thermo* project.

## References

- [1] Y. Chen, S. Deng, X. Xu and M. Chan. A study on the operational stability of a refrigeration system having a variable speed compressor. *International Journal of Refrigeration*, 31(8):1368-1374, May 2008.
- [2] W. Chen, Z. Chen, R. Zhu and Y. Wu. Experimental investigation of a minimum stable superheat control system of an evaporator. *International Journal of Refrigeration*, 25(8):1137-1142, December 2002.
- [3] K.J. Åström and T. Hägglund. Advanced PID Control. *Instrumentation, Systems and Automation Society*, USA, August 2005.
- [4] G. Looye, M. Thümmel, M. Kurze, M. Otter, J. Bals. Nonlinear Inverse Models for Control. In: *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany, 2005.
- [5] C. Richter. Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems. PhD thesis, Technische Universität Braunschweig, 2008.
- [6] C. Schulze. A Contribution to Numerically Efficient Modelling of Thermodynamic Systems. PhD thesis, Technische Universität Braunschweig, 2013.
- [7] M. Gräber, K. Kosowski, C. Richter, W. Tegethoff. Modelling of Heat Pumps with an Object-Oriented Model Library for Thermodynamic Systems. *Mathematical and Computer Modelling of Dynamical Systems*, Taylor & Francis, October 2010.
- [8] T. Hägglund, K.J. Åström. Revisiting the Ziegler-Nichols rules for PI control. *Asian Journal of Control*, 4(4):364-380, 2002.



# Adsorption energy systems library - Modeling adsorption based chillers, heat pumps, thermal storages and desiccant systems

Uwe Bau<sup>1</sup> Franz Lanzerath<sup>1</sup> Manuel Gräber<sup>2</sup> Stefan Graf<sup>1</sup>  
Heike Schreiber<sup>1</sup> Niklas Thielen<sup>1</sup> André Bardow<sup>1</sup>

<sup>1</sup>RWTH Aachen University, Institute of Technical Thermodynamics  
Schinkelstr. 8, 52062 Aachen, Germany

<sup>2</sup>TU Braunschweig, Institute of Thermodynamics  
Hans-Sommer-Straße 5, 38106 Braunschweig, Germany

andre.bardow@itt.rwth-aachen.de

## Abstract

A library for dynamic modeling adsorption based thermal systems like chillers, heat pumps, thermal storages or desiccant units is presented. Adsorption devices can serve a wide range of applications but usually consist of the same basic components. By modeling these basic components, the presented model library allows to investigate any interesting topology. Thereby this adsorption library gives the user the opportunity to design and optimize adsorption systems quickly and efficiently. To demonstrate the flexibility of the library and the accuracy of the simulations, three validated examples are presented: A desiccant unit; a thermal storage; and an adsorption chiller.

*Keywords:* adsorption; simulation; validation; modular; chiller; thermal storage; heat pump; desiccant

## 1 Introduction

Physical adsorption of a fluid on a solid surface is a reversible exothermic process which can be efficiently employed in energy systems. Adsorption phenomena can be used to build a wide variety of thermal devices: e.g. heat driven chillers and heat pumps; high density storages; or desiccant units [1, 2]. Since all these devices enable the use of waste heat or solar heat, they can provide heating and cooling demand more efficiently than conventional devices, and thereby help to reduce CO<sub>2</sub> emissions. To exploit this CO<sub>2</sub> reduction potential, adsorption based thermal devices have

to be well designed. The design of adsorption systems is challenging due to their intrinsic dynamic nature: During operation, adsorption devices switch between ad- and desorption phases, meaning they work discontinuously. In addition, they have a characteristic energy output peak at the beginning. Besides, adsorption devices consist of several components, all being influencing the performance. In an optimal design, these components need to be balanced avoiding oversized components on the one hand and bottlenecks on the other. To meet this optimal design challenge, time-dependent models have been developed to describe and improve adsorption based devices (see e.g. [3, 4, 5]). These models have been developed mainly to study the performance of one specific configuration. To enable the library presented here to describe all kinds of adsorption based thermal devices, a generic modular approach is taken to model the adsorption process. To the best knowledge of the authors only the model of Schicktanz and Núñez describes an adsorption chiller by using a modular approach [6]. By dividing the chiller in its basic components, this approach allows components to be exchanged. Joos et al. presented a modular Modelica library for separation processes including adsorption [7], that can be used to model separation processes, but is not designed for thermal applications, in contrast to the presented library in this paper.

In Section 2, the adsorption phenomenon is described briefly. In Section 3, the library structure and both its basic and enhanced models are described. To illustrate the library, Section 4 contains three examples of validated adsorption systems.

## 2 Adsorption

Adsorption describes the process of attaching fluid molecules to the surface of a (porous) solid, so called sorbent. For a detailed introduction, the reader is referred to Kärger, Ruthven and Theodorou [8]. The internal energy  $u$  of the adsorbed fluid is lower than of the liquid phase and the vapor phase, leading to an exothermic process:

$$u_{ad} < u_l < u_v . \quad (1)$$

This difference between the energies of the liquid and the adsorbed phase is called bond energy. The amount of fluid (adsorbate) adsorbed by the sorbent is described by the loading:

$$w = \frac{m_{ad}}{m_{sor}} , \quad (2)$$

and depends on both the system temperature  $T$  and pressure  $p$ . The relation between loading, temperature, and pressure is described by the thermodynamic equilibrium, which is specific for every working pair. Working pairs (solid / fluid) often used for thermal applications are for example zeolite / water, silica gel / water, and active carbon / methanol. Equilibrium data is used to determine start and end points of adsorption processes. The system dynamics are determined by heat and mass transfer which occur simultaneously: the fluid has to reach a free surface location where it can be adsorbed, while heat has to be transported within the porous solid at the same time. Mass transfer takes place by convective and diffusive processes. Heat transfer can usually be modeled by a mixture of heat conduction in the materials and contact resistances between them.

To control and use the adsorption process in thermal devices, sorbent material and working fluid are heated or cooled in a defined way. For this purpose, either sorbent or working fluid are connected to heat exchangers, building key components of an adsorption device: evaporator; condenser; or adsorber. By connecting the different components in the right way, it is possible to build chillers or heat pumps, thermal storages, as well as air drying units. For more information regarding the design of specific devices see also [9] and [10].

## 3 Library structure

The presented model library follows a modular approach. By coupling a reusable model (working pair

is exchangeable) of the adsorption process to heat exchangers, it enables the user to model a wide range of adsorption based devices.

The presentation in this section follows the library structure shown in Figure 1, by classifying the models as media, basic components, and enhanced components.

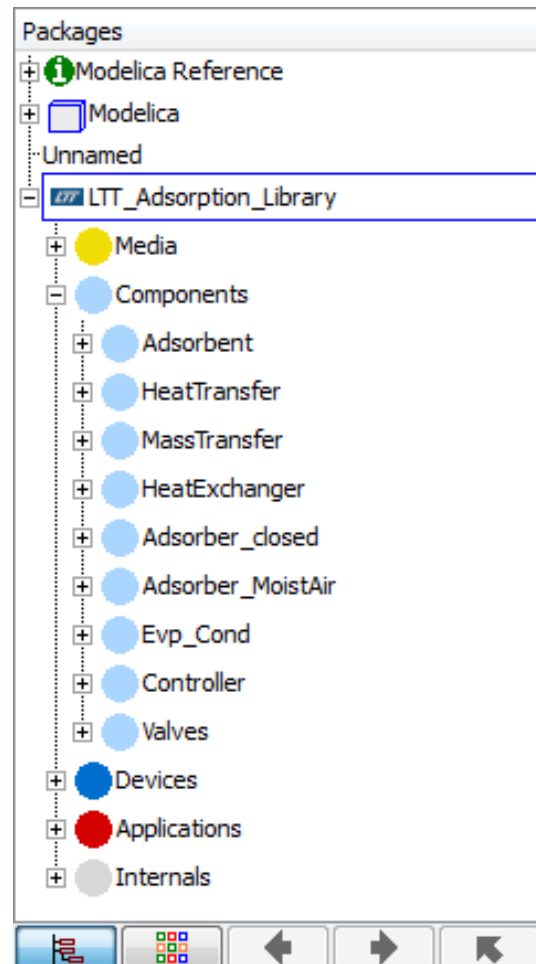


Figure 1: Structure of adsorption library

### 3.1 Media

As described in Section 2, the core of every adsorption based device is the working pair. The media model describes the working pairs characteristic properties: equilibrium data; internal energy of the adsorbed fluid ( $u_{ad}$ ); and specific heat capacity of both sorbent ( $c_{sor}$ ) and adsorbate ( $c_{ad}$ ).

In literature, many approaches describing adsorption equilibria can be found, e.g. the model of Langmuir [11], assuming adsorption only in one layer, the Brunauer, Emmet and Teller model (BET) [12], allowing adsorption in more layers, or the Dubinin model

[13], which is based on Polanyi's potential theory [14]. Which approach is used is not important for the media model as long as adsorbate pressure  $p_{ad}$  at equilibrium state can be described as a function of temperature  $T$  and loading  $w$ :

$$p_{ad} = f(T, w) . \quad (3)$$

The specific energy in the adsorbed state ( $u_{ad}$ ) is derived from the used adsorption model.

The total heat capacity of sorbent and adsorbate  $c_{tot}$  is the sum of the individual heat capacities:

$$c_{tot} = c_{sor} + w c_{ad} . \quad (4)$$

In the present library, most equilibrium data are described using the Dubinin model.

The properties of the fluid are based on TILMedia [15], a library provided by TLK-Thermo GmbH.

### 3.2 Basic components

The following components are the smallest units the library consists of. All extended components and device models are based on these basic components.

#### Adsorbent

In the adsorbent model, equilibrium data, provided by a media model, and mass and energy balances for sorbent and working fluid are combined. The adsorbent is described by a lumped model; i.e., with a homogeneous temperature, loading and pressure distribution. State variables of the model are temperature  $T$  and loading  $w$ . Mass of fluid vapor within the adsorbent can be neglected, in other words, all working fluid within the adsorber is assumed to be in adsorbed state. The adsorbent model contains heat and fluid ports, allowing for heat and mass transfer (see Figure 2).

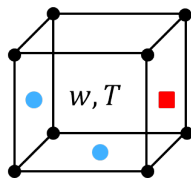


Figure 2: Scheme of adsorbent including the state variables: loading  $w$ ; and temperature  $T$ , as well as the connection ports: fluid (blue); and heat (red)

The mass balance of the working fluid is given by:

$$\frac{dm_{ad}}{dt} = m_{sor} \frac{dw}{dt} = \dot{m}_{fluid,in} - \dot{m}_{fluid,out} . \quad (5)$$

The energy balance is given by:

$$c_{tot} \frac{\partial T}{\partial t} + u_{ad} \frac{\partial w}{\partial t} = \frac{1}{m_{sor}} (\dot{m}_{fluid,in} h_{in} - \dot{m}_{fluid,out} h_{out} + \dot{Q}) . \quad (6)$$

The fluid is leaving the adsorber as vapor with adsorber temperature  $T$  and equilibrium pressure of adsorbent  $p_{ad}$ . The media model used can be changed easily, allowing for a high flexibility. For example, it is possible to investigate the influence on system performance of a changed working pair.

#### Heat transfer

The heat transfer model connects the adsorbent to a heat exchanger or to the environment. The heat flux is described by:

$$\dot{Q} = \alpha A \cdot \Delta T . \quad (7)$$

The implemented heat transfer coefficient  $\alpha$  can be changed. For example, it can be chosen to be a constant parameter or to be dependent on the flow velocity as in the air drying unit described (Section 4).

#### Mass transfer

Mass transfer to or from the adsorbent is described similarly to the heat flow as a linear function of a driving potential (LDF approach). As driving force, either the pressure difference between the pressure on the outside and adsorbate pressure ( $p$  and  $p_{ad}$ ), or the difference between actual and equilibrium loading ( $w_{ad}$  and  $w_{eq}$ ) can be used:

$$\dot{m} = \beta_p (p - p_{ad}) ; \quad (8)$$

or

$$\dot{m} = \beta_w (w_{ad} - w_{eq}) . \quad (9)$$

In this library, both equations are implemented. Since fluid entering the adsorbent is assumed to be adsorbed instantaneously, the mass transfer coefficient  $\beta$  includes not only convective flow resistances but also diffusive resistances occurring within the adsorbent. It is also modeled modular, allowing the user to choose between different models to determine mass transfer coefficient  $\beta$ , similar to the heat transfer coefficient (see also Section 4).

### Tube (heat exchanger)

Heat exchangers for closed adsorption systems are represented by a tube model based on TIL library [15]. The tube model is discretized in flow direction. Heat transfer on the inside is described similarly to equation (7) also using a changeable  $\alpha$ . For liquids, e.g Dittus-Boelter [16] or Sieder-Tate [17] correlations can be used. Also heat conduction within the tube wall is modeled.

### Gas volume (heat exchanger)

An open adsorption system is a system which is not sealed and therefore allows mass transfer between adsorption system and ambient. The ambient, more specifically the surrounding moist air, fulfills two tasks simultaneously: it allows for heat and for mass transfer. Therefore the ambient can be regarded as both, a form of heat exchanger as well as a form of evaporator or condenser. The surrounding air is modeled by a gas volume (see Figure 3). The state variables of the gas volume are gas temperature  $T$ , water mass fraction  $X$ , and air density  $\rho$ . To allow for a gas flow through the volume, the model has two gas ports. It also contains a heat port and a fluid port which are connected to the adsorbent to exchange water (working fluid for open systems) and heat.

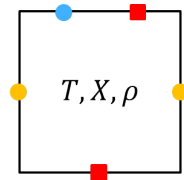


Figure 3: Scheme of gas volume including the state variables: temperature  $T$ ; water mass fraction  $X$ ; and density  $\rho$ , as well as the connection ports: fluid (blue); heat (red); and gas (yellow)

The mass balance of the inert gas component (dry air) is:

$$\frac{dm_{\text{dryair}}}{dt} = \dot{m}_{\text{dryair,in}} - \dot{m}_{\text{dryair,out}} \quad (10)$$

The mass balance of water is correspondingly:

$$\frac{dm_{\text{water}}}{dt} = \dot{m}_{\text{water,in}} - \dot{m}_{\text{water,out}} + \dot{m}_{\text{water,ads}} \quad (11)$$

which can also be written as:

$$m_{\text{air}} \frac{dX}{dt} = \dot{m}_{\text{air,in}} X_{\text{in}} - \dot{m}_{\text{air,out}} X + \dot{m}_{\text{water,ads}} \quad (12)$$

with the water mass fraction:

$$X = \frac{m_{\text{water}}}{m_{\text{air}}} \quad (13)$$

The energy balance of the gas volume is given by:

$$\frac{dU_{\text{air}}}{dt} = \dot{H}_{\text{air,in}} - \dot{H}_{\text{air,out}} + \dot{H}_{\text{water,ads}} + \dot{Q} \quad (14)$$

Water is assumed to leave the gas volume as vapor with gas temperature  $T_{\text{gas}}$  and partial pressure  $p_{\text{water}}$ .

### 3.3 Enhanced components

The main component of an adsorption system is the adsorbent. For closed adsorption systems, also the evaporator and condenser are of major importance. The library already includes prebuild enhanced components, consisting of basic components. By parameter choice these enhanced components can be adapted for many applications.

#### Closed Adsorber

The closed adsorber model consists of an adsorbent model, a heat exchanger model, as well as models for heat and mass transfer (see Figure 4).

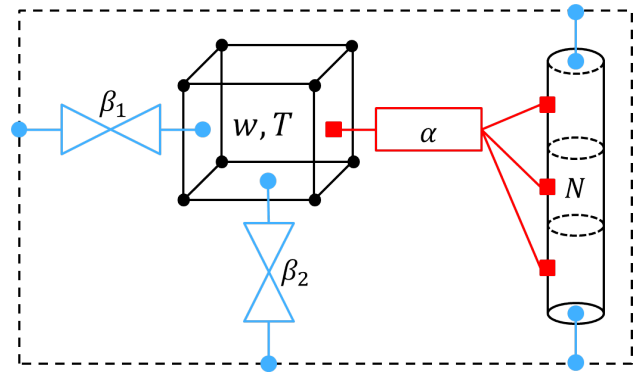


Figure 4: Scheme of closed adsorber model consisting of adsorbent (see Figure 2), heat exchanger (right), mass resistances ( $\beta_1$  and  $\beta_2$ ), and heat resistance ( $\alpha$ )

#### Moist Air Adsorber

The moist air adsorber model consists of an adsorbent volume, a gas volume and a wall volume. These volumes are again connected by models for heat and mass transfer (see Figure 5). The moist air adsorber model can be discretized in flow direction.

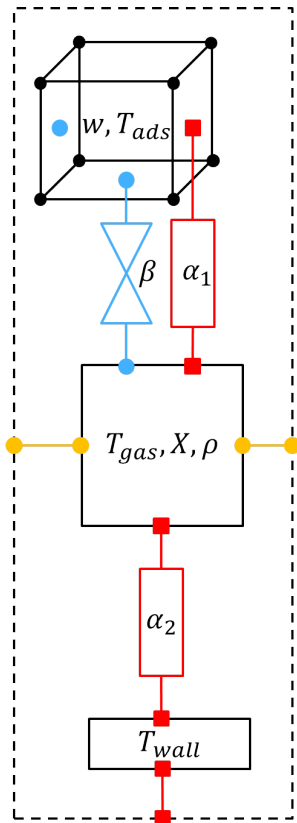


Figure 5: Scheme of moist air adsorber model consisting of an adsorbent (see Figure 2), a gas volume (see Figure 3), a mass resistance ( $\beta$ ), heat resistances ( $\alpha_1$  and  $\alpha_2$ ), and a wall model

## Evaporator / Condenser

The evaporator / condenser model consists of a vapor-liquid-equilibrium (VLE) fluid volume and a heat exchanger. The fluid volume is thermally connected to the heat exchanger and is assumed to be in the two-phase region. The model has a liquid and a vapor port. At the vapor port, the fluid leaves as saturated vapor; at the liquid port it leaves as saturated liquid.

## 4 Validated examples

The library presented in Section 3 is used to model several adsorption based devices. To demonstrate model accuracy, three validated examples are presented: an air drying unit; an adsorption thermal storage; and an adsorption chiller.

### 4.1 Adsorption dryer

An adsorption dryer or desiccant unit is used to simultaneously reduce air humidity and preheat the air. During adsorption, cold humid air enters the dry adsorber.

Until equilibrium state is reached water is adsorbed and the adsorption enthalpy is released. For regeneration of the unit by desorption, hot dry air flows through the adsorber. The air leaves the adsorber with a higher water loading and decreased temperature.

The adsorber model is validated using experimental data by Pesaran and Mills [18]. The used adsorber has a cylindrical shape and is filled with silica gel (Equilibrium data of silica gel is given in [19]).

The driving potential used for mass transfer is pressure difference  $\Delta p$  (equation (8)). Coefficients for mass and heat transfer are modeled by correlations dependent on the Reynolds number based on Hougén and Marshall [20]. Since the model determines the outlet temperature and mass fraction from the inlet values, geometries and input parameters, the model is predictive.

Figure 6 and 7 show temperature and water mass fraction during an adsorption process at adsorber inlet and outlet.

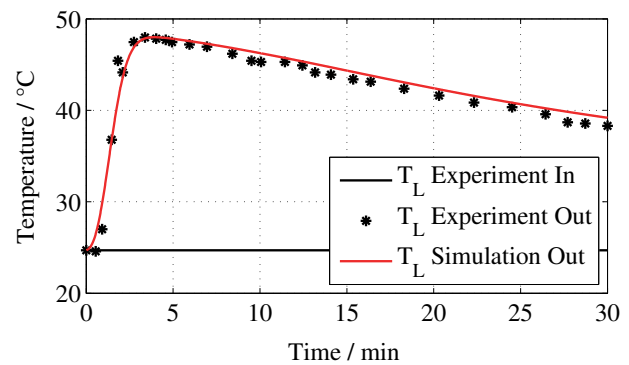


Figure 6: Air temperature  $T$  at inlet and outlet during adsorption

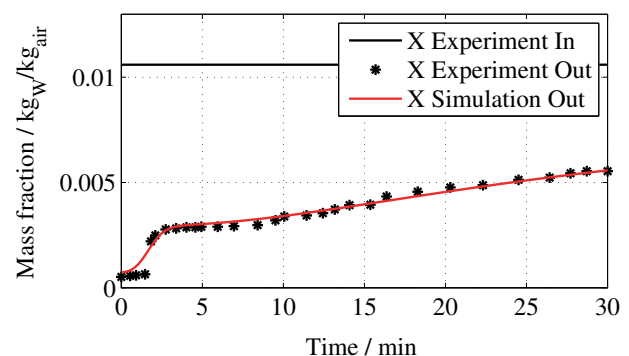


Figure 7: Water mass fraction  $X$  at inlet and outlet during adsorption

Validation shows good agreement in both, temperature  $T$  and mass fraction  $X$ . The input conditions are

changed suddenly at  $t = 0$ s, resulting in a step function. The output is reacting by an increase in outlet temperature due to adsorption heat emitted. Around 5 min after start, temperature peaks at  $46.9^\circ\text{C}$  and afterwards declines only slightly. By reaching a temperature of  $45.4^\circ\text{C}$ , simulation corresponds well to the measured peak after around 5 min. Water mass fraction between inlet and outlet is reduced by about  $0.01 \text{ kg}_{\text{water}}/\text{kg}_{\text{air}}$  with nearly dry air at the outlet. Measured and simulated water loading fit almost perfectly. Pesaran and Mills [18] have experimentally varied input parameters, adsorber length, and particle size of adsorbent grains in their experiments. For all six tested input variations simulation and the experiments agreed well (not shown).

## 4.2 Thermal adsorption storage

Thermal energy, stored in an adsorption storage, can be divided into a sensible and a latent part. Sensible heat increases with system temperature, while at the same time, latent heat is stored by desorption of water. The desorbed water leaves the adsorber as vapor and is condensed afterwards in the condenser. During condensation, low grade heat is emitted at medium or low temperature level, that can be either used or emitted to the ambient. When using the stored heat (discharging the storage), water is evaporated using low grade heat. The evaporated water flows to the adsorber where it is adsorbed and heat at process temperature is released.

The storage model is validated using experimental data from a storage prototype at our institute [21]. Figure 8 illustrates the thermal adsorption storage model. Since temperature difference between adsorber and evaporator is high and there is no insulation between these components, they are thermally connected in the model. For further details regarding the experimental setup see Binkert [22].

The working pair used in the experiments is zeolite 13X / water. Equilibrium data for this pair can be found in Núñez [23]. Heat and mass transfer coefficients were fitting experimental and simulation data. Optimization criterion used to determine fit accuracy is the root mean square deviation (RMSD) of adsorber heat. Figure 9 shows the adsorber heat flow during desorption, Figure 10 during adsorption. Desorption temperature increases to almost  $200^\circ\text{C}$ , with a condensation temperature of  $90^\circ\text{C}$ . After around 3 hours, the operation mode is switched from desorption to adsorption (loading to unloading). Adsorption takes about 30 min with adsorption temperature falling from  $200^\circ\text{C}$  to  $120^\circ\text{C}$ . Evaporation is kept at  $60^\circ\text{C}$  during

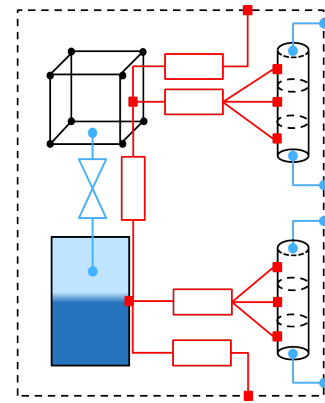


Figure 8: Scheme of thermal adsorption storage consisting of an adsorbent (see Figure 2), two heat exchangers (right), mass resistances, heat resistances, and an evaporator / condenser model

adsorption. Although time varies by factor 6 and heat flow even by factor 10, comparison between experimental data and simulations shows good agreement for both, desorption and adsorption. During desorption, the peak in heat flow at 15 min is slightly overestimated by the simulation, for adsorption it can be observed that the trend fits well. At the end of adsorption phase, slight differences between experiment and simulation occur, but they are still in a tolerable range.

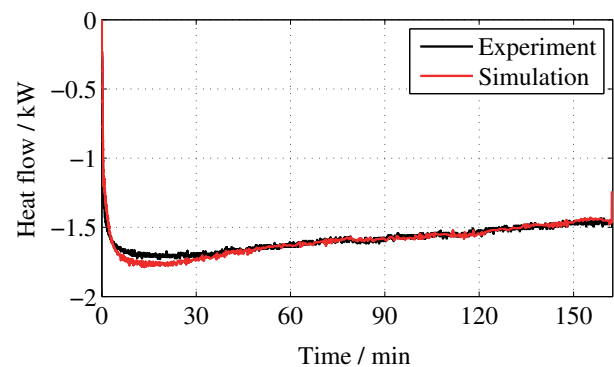


Figure 9: Heat flow  $\dot{Q}$  of adsorber during desorption

## 4.3 Adsorption Chiller

The investigated adsorption chiller consists of an adsorber, an evaporator, and a condenser. These components are separated by valves, allowing to disconnect the adsorber from the evaporator and condenser separately. Because adsorption and desorption phases take place successively (Section 1), an adsorption chiller with only one adsorber bed operates discontinuously. During the adsorption phase, water is evaporated at a low temperature level (cooling output). The water



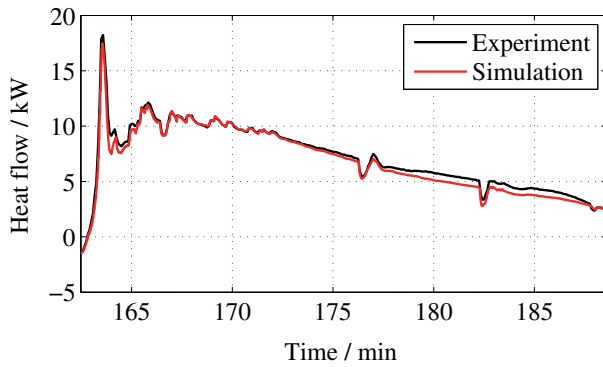


Figure 10: Heat flow  $\dot{Q}$  of adsorber during adsorption

flows to the adsorber where it is adsorbed and heat is emitted. Thus, the adsorber has to be cooled, which occurs at a medium temperature level. During the desorption phase, the adsorber bed is regenerated. By using the heat exchanger, heat is passed into the adsorber at a high temperature level. The desorbed water vapor flows to the condenser, where it is condensed at a medium temperature level again. The high temperature level is determined by the heat source available: mostly waste or solar heat. The medium temperature level is usually determined by the ambient temperature; and the low temperature level by the cooling application. An experimental setup of the described adsorption chiller at our institute is used for model validation. Figure 11 shows a schematic picture of the model, representing the experimental setup.

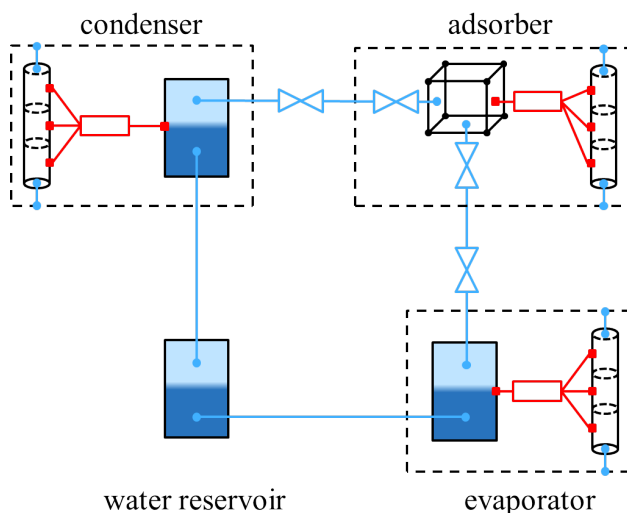


Figure 11: Scheme of adsorption chiller consisting of an adsorbent (see Figure 2), evaporator, condenser, mass resistances, heat resistances, and a water reservoir model

The model was investigated for both silica gel and zeolite 13X. In this paper we show validation for silica gel. A more in-depth validation can be found in Lanzerath [24], discussing the effects of varying input parameters and changed adsorbent materials. Equilibrium data for silica gel are implemented using the Dubinin approach from Schawe [25].

Coefficients for heat and mass transfer are used as fitting parameters. They are determined by using the experimental data of a complete cycle (adsorption and desorption) and minimizing the RMSD of heat flows between experimental and simulation data. The fitted heat and mass transfer parameters were kept constant for all variations of experimental settings and proved to be robust [24]. Figure 12 shows the adsorber heat flow, Figure 13 the heat flows of evaporator and condenser. Although only lumped models are used for adsorber, evaporator and condenser, it can be observed, that system dynamics, as well as steady state conditions, are predicted well.

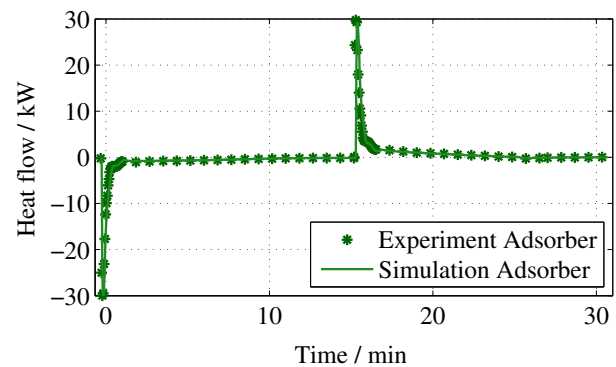


Figure 12: Heat flow of adsorber

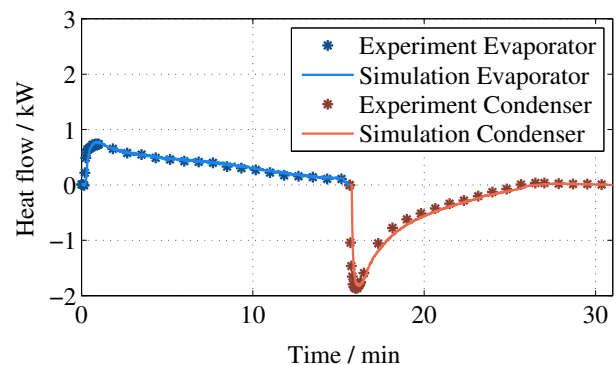


Figure 13: Heat flows of evaporator and condenser

## 5 Conclusion

An adsorption library, capable of modeling a wide variety of adsorption devices is presented. The models were validated using three examples: a desiccant system; an adsorption thermal storage; and an adsorption chiller. All examples show very good agreement of simulation results with experimental data. Thereby, the library is shown to be a valuable toolbox for research purposes, allowing to describe, analyze and improve the design and topology of adsorption based devices.

## References

- [1] Meunier F E. Adsorption heat powered heat pumps. *Applied Thermal Engineering*, 61, pp. 830-836, 2013.
- [2] Daou K, Wang R Z, Xia Z Z. Desiccant cooling air conditioning: a review. *Renewable and Sustainable Energy Review*, 10, pp. 55-77, 2006.
- [3] Douss N, Meunier F E, Sun L-M. Predictive Model and Experimental Results for a Two-Adsorber Solid Adsorption Heat Pump. *Industrial and Engineering Chemistry Research*, 27, pp. 310-316, 1988.
- [4] Maggio G, Freni A, Restuccia G. A dynamic model of heat and mass transfer in a double-bed adsorption machine with internal heat recovery. *International Journal of Refrigeration*, 29, pp. 589-600, 2006.
- [5] Wang X, Chua H T. Two bed silica gel-water adsorption chillers: An effectual lumped parameter model. *International Journal of Refrigeration*, 30, pp. 1417-1426, 2007.
- [6] Schick Tanz M, Núñez T. Modelling of an adsorption chiller for dynamic system simulation. *International Journal of Refrigeration*, 32, pp. 588-595, 2009.
- [7] Joos A, Dietl K, Schmitz G. Thermal Separation: An Approach for a Modelica Library for Absorption, Adsorption and Rectification. In: *Proceedings 7th Modelica Conference*, 2009.
- [8] Kärger J, Ruthven D M, Theodorou D N. Diffusion in Nanoporous Materials. *Wiley-VCH*, 2012.
- [9] Wang D, Zhang J, Tian X, Liu D, Sumathy K. Progress in silica gel-water adsorption refrigeration technology. *Renewable and Sustainable Energy Reviews*, 30, pp. 85-104, 2014.
- [10] Misha S, Mat S, Ruslan M H, Sopian K. Review of solid/liquid desiccant in the drying applications and its regeneration methods. *Renewable and Sustainable Energy Reviews*, 16, pp. 4686-4707, 2014.
- [11] Langmuir I. The Adsorption of Gases on Plane Surfaces of Glass, Mica and Platinum. *Journal of the American Chemical Society*, 40, pp. 1361-1403, 1918.
- [12] Brunauer S, Emmett P H, Teller E. Adsorption of Gases in Multimolecular Layers. *Journal of the American Chemical Society*, 60, pp. 309-319, 1938.
- [13] Dubinin M M. Adsorption in micropores. *Journal of Colloid and Interface Science*, 23, pp. 487-499, 1967.
- [14] Polanyi M. Adsorption of gases by a non-volatile adsorbent. *Verhandlungen der Deutschen Physikalischen Gesellschaft*, 18, pp. 55-80, 1916.
- [15] Gräber M, Kosowski K, Richter C, Tegethoff W. Modelling of heat pumps with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16, pp. 195-209, 2010.
- [16] Dittus W, Boelter L M K. Heat transfer in automobile radiators of the tubular type. *University of California - Publications in Engineering*, 2, pp. 443 - 461, 1930.
- [17] Sieder E N, Tate G E. Heat Transfer and Pressure Drop of Liquids in Tubes. *Industrial and Engineering Chemistry*, 28, pp. 1429-1435, 1936.
- [18] Pesaran A A, Mills A F. Moisture transport in silica gel packed beds II. Experimental study. *International Journal of Heat and Mass Transfer*, 6, pp. 1051-1060, 1987.
- [19] Pesaran A A, Mills A F. Moisture transport in silica gel packed beds I. Theoretical study. *International Journal of Heat and Mass Transfer*, 6, pp. 1037-1049, 1987.

- [20] Hougen O A, Marshall W R. Adsorption from a Fluid Stream flowing through a stationary Granular Bed. *Chemical Engineering Progress*, 46, pp. 197-208, 1947.
- [21] Schreiber H, Graf S, Lanzerath F, Bardow A. Adsorption heat storage for combined heat and power units in industrial batch processes. *International Sorption Heat Pump Conference*, 2014.
- [22] Binkert J, Lauer J, Diaconu A, Ruß W, Schreiber H, Bardow A. Entwicklung einer Verfahrenskombination aus Zeolithwärmepumpe, Vakuumeindampfsystem und Blockheizkraftwerk zur energieeffizienten Wärmeversorgung von Brauereien. *Deutscher Bund für Umwelt und Naturschutz (DBU)*, 2013.
- [23] Núñez T. Charakterisierung und Bewertung von Adsorbentien für Wärmetransformationsanwendungen. *PhD thesis, Albert-Ludwigs-Universität Freiburg*, 2001.
- [24] Lanzerath F, Seiler J, Bau U, Bardow A. A modular experimental and simulation approach for the systematic development of adsorption heat pumps. *International Sorption Heat Pump Conference*, 2014.
- [25] Schawe D. Theoretical and Experimental Investigations of an Adsorption Heat Pump with Heat Transfer between two Adsorbers. *PhD thesis, Universität Stuttgart*, 2001.



# A new Implementation of the N-D Lookup Tables

Torsten Sommer

Markus Andres  
Modelon GmbH

Stephan Diehl

Agnes-Pockels-Bogen 1  
D-80992 Munich, Germany

torsten.sommer@modelon.com

markus.andres@modelon.com

stephan.diehl@modelon.com

## Abstract

The HDF5Table library is an open-source solution for the efficient handling, exchange and interpolating access of typical data sets in system simulation. The library consists of C-functions, python scripts and examples and can be used with different applications like Modelica or Simulink. Furthermore a comprehensive set of tools that allows the user to create, migrate, edit, compare and manage the datasets is available.

The application range covers data import from measurements or other simulations, integration of datasets in preprocessing routines, the usage of the datasets in the simulation and the post processing of simulation results. To eliminate a major source of errors after data exchange between simulation tools or different companies and to validate the datasets each dataset can have a physical unit and quantity attached to it. The table data can be easily accessed with different methods for inter- and extrapolation. To persist and exchange the data sets a subset of the HDF5 standard is used. With the HDF5 API the data access is fast for large files with many variables containing millions of values and the datasets can be opened in many other tools.

*Keywords: HDF5; lookup tables; unit and quantity safety; interpolation; extrapolation*

## 1 Introduction

Lookup tables traditionally play a major role in industrial simulations. They are used in a wide range of applications where physical models or parameters are not available or the evaluation of the existing models is computationally too expensive. Real-time simulations and hardware-in-the-loop setups are two prominent examples. In these systems lookup tables are used to re-play recorded stimulus from measurements as well as pre- and post-processed data from

test benches. Another application is pre-calculated lookup tables from long running system or finite element simulations.

A number of solutions exist for Modelica and other simulation platforms some of which are discussed in detail in the following section. All of these solutions suffer from different limitations and problems the proposed implementation together with a set of supporting tools is trying to solve.

## 2 Existing Solutions

The Modelica Standard Library (MSL) provides a number of tables in its Blocks Package. A general separation is done based on the provided input functionality. Here the prefix Combi shows the capability of the table to either take direct user input in the form of parameters from the Modelica environment or to read data from files. In industrial applications the second option is nearly solely used as it is more convenient even for tables of modest size.

Table blocks for different needs are provided and therefore are split into the Blocks.Sources and the Blocks.Tables package. The separation into the packages is done as the tables in the Sources package implicitly define the single input of the table as the simulation time. These tables are mainly used for the playback of recorded continuous stimulus e.g. steering input or velocity signals over time. As these tables have a single input and are well suited for time series simulation they are not in the focus of this paper.

This leads to the Blocks.Tables package. These tables are often used to cover effects that cannot be modeled based on physics with reasonable effort or performance. Therefore tables are used to approximate the behavior depending on a varying number of inputs. Still for the tables in Blocks.Tables the number of inputs is limited to one or two which becomes a limiting factor in many applications. Therefore Dymola provides the DataFiles package that contains the TableND which extends the number of inputs to

a theoretically unlimited amount. As indicated by the missing prefix, data can only be read from .mat files. The DataFiles Package contains a number of functions that support the user with the generation of the necessary .mat files. Additionally some Matlab scripts are provided to support the data storing process.

Now at first glance for Dymola users there is the possibility to have capable tables in their simulations using either the MSL's Tables package or the DataFiles package. Still there are a couple of limitations to that which we want to point out now. The TableND's data format is limited to version 4 .mat files which limits the names in the file to 19 characters and – much more of interest here – limits the number of possible dimensions stored in the files to two. Therefore the multi-dimensional tables cannot be stored in their natural format but have to be converted. This problem is overcome by generating three vectors by convention named `dim`, `grid` and `table`. The length of the vector `dim` equals the number of dimensions of the table each entry indicating the amount of scale values in that dimension. In the variable “`grid`” the scales for all dimensions are stored implying that the length of `grid` has to be equal to the sum of the elements in `dim`. Finally in the table variable finally all values of the multidimensional table are stored in one vector. All of this makes it very hard to interpret the values in the table without suitable conversion scripts.

Another disadvantage of this solution is the different behavior of CombiTables and TableND when it comes to extrapolation. Whereas the CombiTables extrapolate linearly the TableND keeps values constant when exiting the defined area. For version 3.2.1 of the MSL efforts have been completed to make the implementation of the MSL tables open source and to enhance their functionality. New features include that outputs of tables can now be differentiated once, and newer .mat File formats are supported [7]. Still that does not have any influence on the limiting factors mentioned before, as this is only valid for the tables contained in the MSL.

### 3 Goals of the new Implementation

The goal of the proposed implementation is to come up with a library that provides a superset of the features of the existing solutions that is entirely open source and accompanied by comprehensive set of tools to leverage its functionality and ease the transition from existing solutions. The scope of possible applications for both tables and tooling goes beyond Modelica models since the tables can be easily port-

ed to other simulation platforms (e.g. Simulink, CarMaker) or integrated into custom C-based solutions. This takes the concept of separation of model and data to a higher level as data can be used independently from the simulation tool.

Furthermore one of the basic concepts of Modelica, namely unit and quantity safety is adapted for table-based data. Every dataset regardless of its dimensions can have a physical unit and quantity (and other attributes) attached to it which allows the table block to validate the data upon loading and eliminates a major source of errors that arises from incompatible data being used in the simulations.

Future versions of the library will also be able to record signals from a simulation and provide support for ‘compiled-in’ data which is especially useful for platforms that do not have a file system or to protect the intellectual property contained in the data.

## 4 Features

To provide a one-stop solution for all uses of table based data the table block features different methods to inter- and extrapolate the sampled data which are briefly presented in the following sections.

### 4.1 Interpolation

Generally two major uses of tables in models can be distinguished: the first is the playback of recorded continuous stimulus e.g. steering input or discrete data like bus events. Both types are usually one-dimensional and have the time as abscissa.

The second major use case for tables is found in models that cannot be physically modeled and thus use measured data to approximate their behavior or are too complex to be simulated e.g. in a real-time context. These table-based models usually depend on more than one parameter and thus require multi-dimensional tables. A prominent example in which both types of tables can be found are hardware-in-the-loop test benches.

The HDF5Table block provides three interpolation methods that can be configured on a per-instance basis: ‘Hold’, ‘Linear’ and ‘Akima’.

‘Hold’ will simply return the previous value in the respective dimension and can be used to playback time discrete data where interpolation does not make sense e.g. the selected gear in a vehicle.

‘Linear’ interpolates linearly between the neighboring sample points in each dimension.

‘Akima’ uses a spline based interpolation method proposed by Akima [5] where the interpolated func-

tion passes through the sample points and its first derivative is continuous.

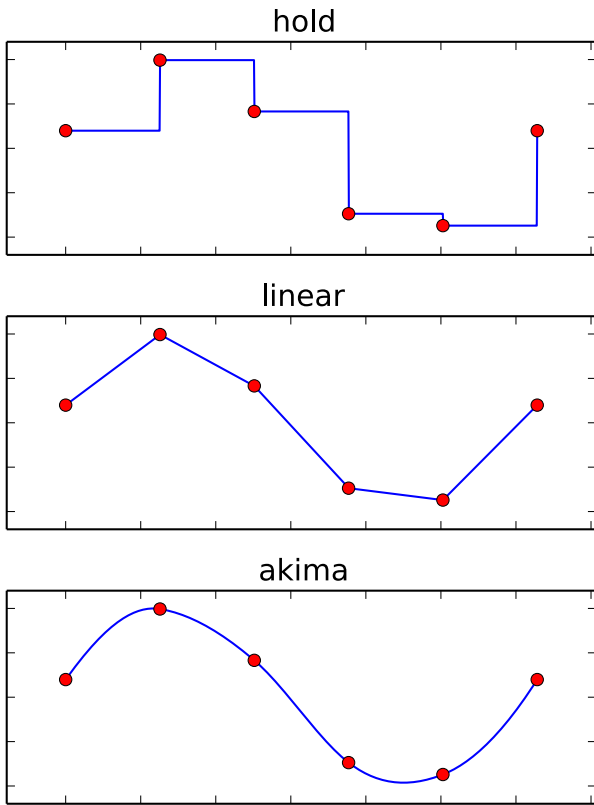


Figure 1 Interpolation Methods

This has a positive impact on the stability of models that use the derivative in their calculations. Figure 1 shows the three interpolation methods applied to a one-dimensional dataset with six equally spaced sample points of a sine function.

#### 4.2 Extrapolation

Five methods are provided for extrapolation:

- Hold
- Linear
- Loop
- PingPong
- None

Similar to the interpolation ‘Hold’ simply repeats the last sample value in the respective dimension. ‘Linear’ uses the last two samples to linearly extrapolate. For the playback of time series it is often desired to infinitely repeat a given set of samples. ‘Loop’ repeats the recorded signal using the selected interpolation method.

The ‘PingPong’ method works similarly to the loop method but instead of starting over it goes back and forth along the respective axis. This method is especially useful for lookup tables used for devices that

have a symmetric characteristic field like electric machines. To better understand this method consider the following example of the loss characteristics of a permanent magnet synchronous machine (PMSM) given as the power losses versus rotational speed as the abscissa and the torque as the ordinate. Three different types of lookup tables are common for PMSMs: first quadrant, first and second quadrant and all four quadrants corresponding to the motor, motor and generator and motor and generator for positive and negative rotation directions. With the ‘PingPong’ method one table block can be used for all three types of tables without conditional instantiation or additional logic inside the model. Figure 2 shows the extrapolated curve of the five methods for a one-dimensional dataset with four equally spaced and linearly increasing samples. The extrapolation method “None” disables extrapolation and raises an error when requested value is outside the range of the table.

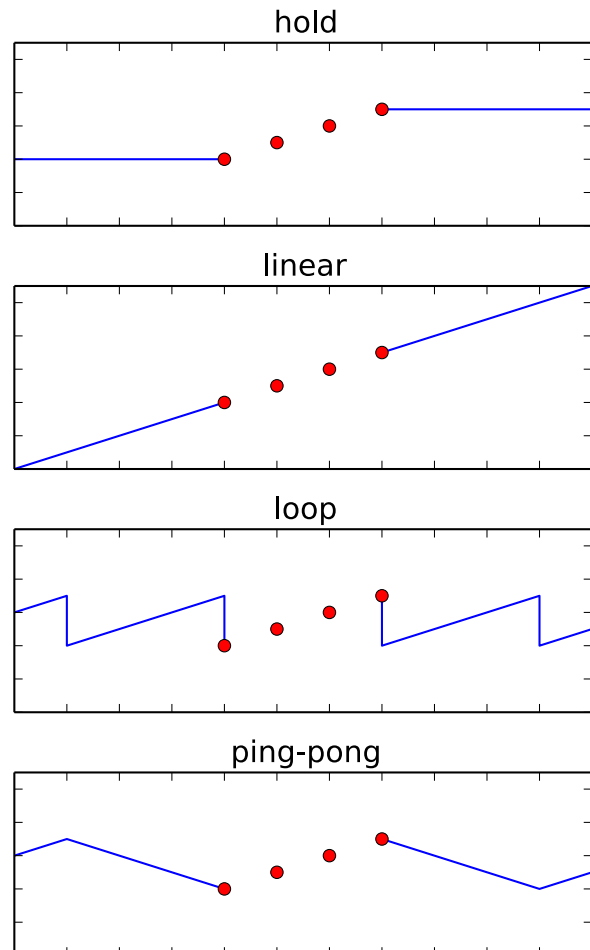


Figure 2 Extrapolation Methods

## 5 HDF5 File Structure

In order to efficiently store and exchange the data for the tables a subset of the HDF5 standard is used. Every table is stored as n-dimensional dataset of type Float64. The scales for each dimension are stored as separate datasets. For every dimension one scale dataset is attached to the table dataset using the HDF5 dimension scales API [3].

To store metadata like the physical units and quantities a set of attributes has been defined that is evaluated when reading the datasets. The following table lists the attribute names together with the required data type and an example value.

Attribute Name	Type	Example
QUANTITY	String	“AngularVelocity”
UNIT	String	“rad/s”
DISPLAY_UNIT	String	“1/min”

QUANTITY and UNIT denote the physical quantity and unit of the stored data. DISPLAY\_UNIT is the unit that is used to display the values to the user e.g. when editing them. The literals used to represent the units and quantities are the same as in Modelica and can be found in the Modelica.SIunits and Modelica.SIunits.Conversions.NonSIunits packages of the MSL.

Scales must always be one-dimensional and strictly monotonic increasing and there must be at most one scale attached per dimension whose length matches the extent of the dataset in the respective dimension. For a three-dimensional dataset with extent 2×3×4 the scale for the second dimension must have exactly three values.

Users may add custom attributes and datasets to store additional metadata, documentation or results and use groups [6] to structure the datasets.

## 6 Modelica

The HDF5Table Modelica library comes as a .mo file that contains the blocks, functions and examples together with the C header files and pre-compiled object libraries for the table and HDF5 as a ready-to-use package. The Modelica functions and blocks are presented in detail in the following sections.

### 6.1 Functions

To directly read and write scalars, vectors and matrices from and to HDF5 files the library includes the following functions that are similar to the readMAT\*

/ writeMAT\* functions in the DataFiles package that ships with Dymola:

- writeVector
- writeMatrix
- attachScale
- readScalar
- readScalarChecked
- readVector
- readVectorChecked
- readMatrix
- readMatrixChecked
- attachScale

The “checked” versions of the functions take the expected unit and quantity as an additional parameter and return an error code if the parameters do not match the unit or quantity stored in the loaded dataset.

The attachScale function attaches a dataset as the scale for the dimension of another dataset. All functions are realized as external C code that can be linked into the model or be called directly using the simulation tool as shown in the figure below.

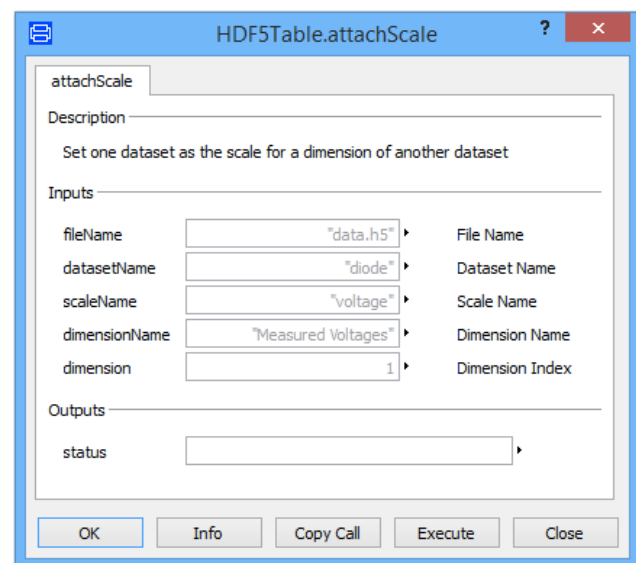


Figure 3 attachScale Function Dialog

The library can also read / write higher dimensional arrays (up to 32 dimensions). Entering higher dimensional arrays however can be cumbersome, as e.g. Dymola does not provide UI support for editing values with more than two dimensions.

### 6.2 Blocks

The core of the library is the NDTable block that loads and interpolates the data during the simulation. It takes the file name of the HDF5 file, the dataset



name and optionally the expected units and quantities of the dataset and scales as input and returns the inter- or extrapolated value for every time step. Figure 4 shows the parameters of the NDTable block with all quantities and units set for a dataset that holds the power losses of an electric machine as a function of rotational speed, torque and source voltage.

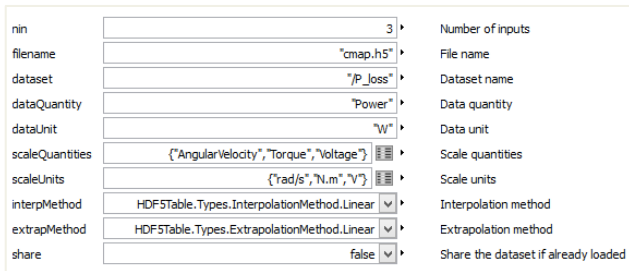


Figure 4 Parameters of the NDTable block

The DatasetRecorder block takes the scales as parameters and sweeps successively over the volume spanned by the scales by applying the actual scale values to the vector output. When a rising edge is detected on the trigger port it records the value on the in-port and applies the next set of scale values to the input until all samples in the volume have been recorded. Using this block tables with a large number of sample points can be generated without the overhead of restarting the simulation which can save a considerable amount of time.

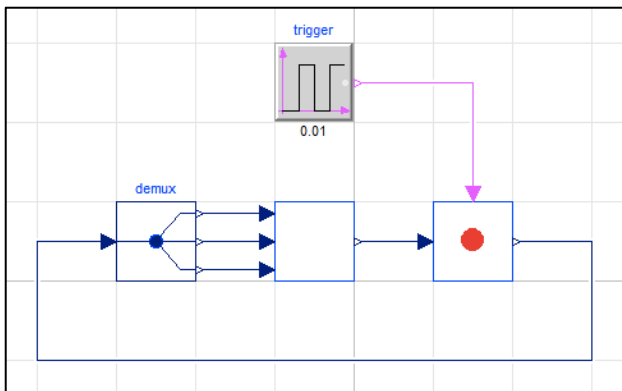


Figure 5 RecorderBlock Example Model

Figure 5 shows the RecordTable example model where the DatasetRecorder block is connected to a dummy model that has three in-ports and whose output is recorded upon every rising flank of the trigger.

## 7 Matlab / Simulink

Matlab and Simulink are two of the most commonly used simulation and scripting environments. Therefore the HDF5Table library also includes a Simulink

S-Function and Matlab scripts that allow users to reuse their existing datasets without changes on this platform. The Simulink table block has the same inter- and extrapolation methods as the Modelica library and the underlying S-Functions are based on the same C-sources which can be a major advantage when porting models to this platform that rely on the specific behavior of the implementation. Just like the Modelica library the Simulink library ships as a pre-compiled shared library that can be instantly used in Simulink.

## 8 Tooling

To leverage the tables in the simulation they are accompanied by a comprehensive set of tools that allow the user to create, migrate, edit, compare and manage the datasets. All tools are included in an Eclipse distribution that is used as an integration platform but can also be obtained and used separately.

The tools include a Python library based on matplotlib [8] and NumPy [9] to read, write, manipulate and plot data that can be debugged and run directly from within Eclipse using the PyDev environment [10]. Python's support for a large number of data formats allows the user to write import and export scripts for their existing data with minimal effort based on the provided examples. With the included Python scripts both the text based and binary data files used by the Blocks.Tables and Datafiles blocks can be converted to HDF5.

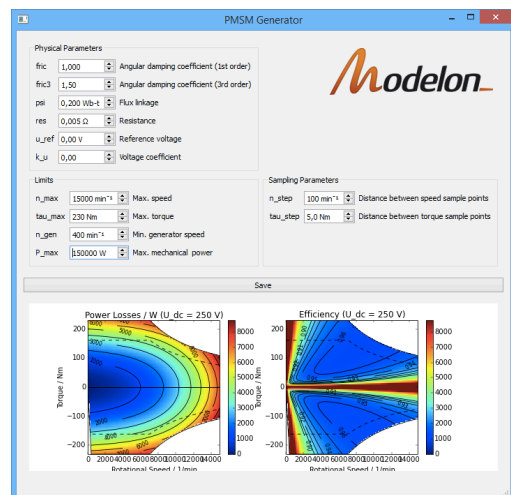


Figure 6 Python script with graphical UI

Figure 6 shows an example Python script with a graphical user interface that allows the user to create characteristic maps of electric machines for use with the NDTable block and that can serve as a basis for custom HDF5 generators or processors.

Furthermore an EMF [11] based editor is included that allows the user to conveniently view, edit and validate the data files. Finally the datasets can be compared and merged with the included EMF compare editor [12].

The figure below shows the HDF5 editor viewing the content of a three-dimensional characteristic map dataset with scales.

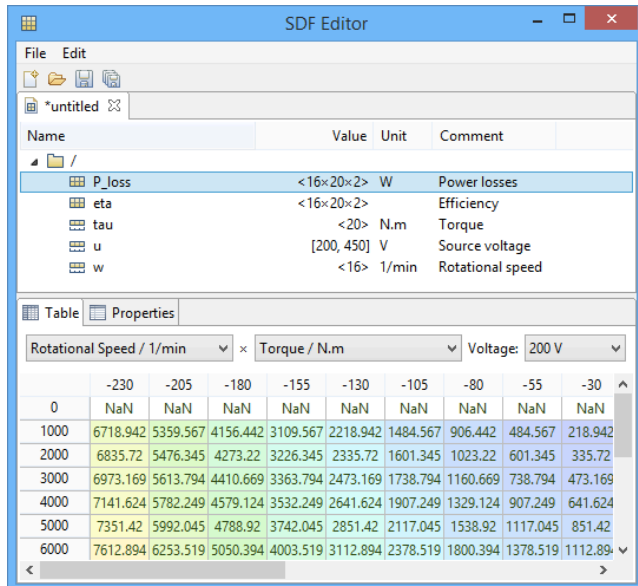


Figure 7 HDF5 Editor

In the editor values are displayed in “display units” and can be conveniently entered as expressions like “sin(pi/4)” to get the sine of a 45 degree angle or “ones(2, 3)” to get a 2×3 matrix of ones. All operations performed in the editor like copy & paste, move, delete etc. can be undone with the standard shortcuts or menu items. Drag & drop is supported to move datasets between groups and even between files.

## 9 Future Work

Currently the library is developed in Visual Studio 2010 for Windows. It is planned to port the library to the hardware and software platforms listed below.

Hardware Platforms:

- Windows 32 and 64 bit
- Linux 32 and 64 bit
- dSPACE SCALEXIO

- dSPACE DS1006

Compilers:

- Microsoft compilers (VC6 and  $\geq$  VS2005 (Win32 and x64))
- MinGW (GCC 4.4.0 and GCC 4.7.2)
- Cygwin (GCC 4.3.0)
- GCC 4.x on Linux

We are also planning to further extend the tooling and to include a comprehensive documentation and examples that showcase common uses and best practices. A future version of the library will include additional inter- and extrapolation methods (e.g. boundary slope extrapolation for the Akima method) and support for the Der() derivative of the interpolated values to reduce simulation time and improve accuracy [1].

## 10 Conclusions

The implemented library and its extensions show that most features of the existing tables in the MSL’s Blocks.Tables and the DataFiles package can be combined in a single table simplifying the application for the user. Additionally the error-proneness of the overall simulation process can be reduced substantially based on unit and quantity checks.

The use of open standards like HDF5 and Modelica guarantee that this open-source implementation is expandable and can be customized for different needs including wider tool support. To enable fast adaption of the presented library a set of tools is provided enabling the user to quickly generate new or migrate existing datasets.

## References

- [1] Modelica Language Specification Version 3.3, Section 12.7, <https://www.modelica.org/documents/ModelicaSpec33.pdf>
- [2] Call for Quotation of an Open Source Implementation of the MSL Table Interpolation Blocks, [https://www.modelica.org/news\\_items/call-texts-to-improve-modelica-2012/2012-12-20-Call-for-quotation-for-MSL-tables.pdf/at\\_download/file](https://www.modelica.org/news_items/call-texts-to-improve-modelica-2012/2012-12-20-Call-for-quotation-for-MSL-tables.pdf/at_download/file)
- [3] HDF5 Dimension Scale API Reference, [http://www.hdfgroup.org/HDF5/doc/HL/RM\\_H5DS.html](http://www.hdfgroup.org/HDF5/doc/HL/RM_H5DS.html)
- [4] Proposal for a Standard Time Series File Format in HDF5, [http://www.bausch-gall.de/ecp12076495\\_PfeifferBausch-GallOtter.pdf](http://www.bausch-gall.de/ecp12076495_PfeifferBausch-GallOtter.pdf)
- [5] "A new method of interpolation and smooth curve fitting based on local procedures", Journal of ACM 17, 4 (1970), 589-602
- [6] HDF5 Group Interface, [http://www.hdfgroup.org/HDF5/doc/RM/RM\\_H5G.html](http://www.hdfgroup.org/HDF5/doc/RM/RM_H5G.html)
- [7] Modelica Newsletter by Martin Otter <https://www.modelica.org/publications/newsletters/2013-2#item227>
- [8] matplotlib, <http://matplotlib.org/>
- [9] NumPy, <http://www.numpy.org/>
- [10] PyDev, <http://pydev.org/>
- [11] Eclipse Modeling Framework, <http://www.eclipse.org/modeling/emf/>
- [12] EMF Compare, <http://www.eclipse.org/emf/compare/>



# Remarks on the Implementation of the Modelica Standard Tables

Thomas Beutlich   Gerd Kurzbach   Uwe Schnabel  
ITI GmbH  
Schweriner Straße 1, 01067 Dresden, Germany  
{beutlich, kurzbach, schnabel}@itisim.com

## Abstract

This article reveals some implementation details regarding the C code of the revised table interpolation blocks released with the Modelica Standard Library (MSL) 3.2.1. The emphasis is placed on the unique features of the `CombiTimeTable` which are the discontinuities by time events and the periodic extrapolation. Basic information on the interpolation by Akima splines and the available table array memory optimization options are mentioned.

*Keywords:* univariate interpolation, bivariate interpolation, periodic extrapolation, time events, Akima splines

## 1 Introduction

For many years there has been no (open source) implementation of the table interpolation blocks of the MSL. Thus, Modelica simulation tools either did not support the table blocks or needed to provide a custom implementation that could lead to different simulation results. One objective of releasing a backward compatible MSL 3.2.1 was to provide an open source implementation of the table blocks based on the Modelica external object interface. This table implementation was named *Modelica Standard Tables* and comprises the following four blocks for univariate and bivariate interpolation

- `Modelica.Blocks.Sources.CombiTimeTable`,
- `Modelica.Blocks.Tables.CombiTable1D`,
- `Modelica.Blocks.Tables.CombiTable1Ds` and
- `Modelica.Blocks.Tables.CombiTable2D`.

The C header and source files of the Modelica Standard Tables are publicly available from <https://svn.modelica.org/projects/Modelica/trunk/Modelica/Resources/C-Sources>.

`//svn.modelica.org/projects/Modelica/trunk/Modelica/Resources/C-Sources`. The C interface functions all start with prefix `ModelicaStandardTables_`. The constructors and destructors of the external table objects have suffix `_init` and `_close`, respectively. If the table data is to be read from an ASCII text file or a MATLAB MAT-File, an interface function with suffix `_read` is used, i.e. file I/O is not part of the construction of the external table object. The actual interpolation functions are labeled by trailing `_getValue` and `_getDerValue` for the interpolation function and the interpolated derivatives, respectively.

## 2 CombiTimeTable

The block `Modelica.Blocks.Sources.CombiTimeTable` is different from standard univariate interpolation since discontinuities (by time events) and periodic extrapolation are considered. For instance, periodic and discontinuous signals like saw-tooth or square-wave w.r.t. simulation time can be modeled in a very convenient and compact way.

### 2.1 Time events

Time events always occur at the table boundaries ( $t_{min}$  and  $t_{max}$ ) of the sample points, i.e. if interpolation switches to extrapolation.

- In case of linear interpolation, additional time events can be modeled by repetition of sample points in the table array. It is guaranteed that there are no time events at interval boundaries with a simple sample point only. Thus the time coordinates are not required to be strictly increasing but monotonically increasing.
- In case of interpolation by constant segments, every interval boundary (defined by the sample

points) leads to a time event, whether or not there is an actual discontinuity in the ordinate values. The time coordinates are not required to be strictly increasing but monotonically increasing. In fact, repeated sample points are not appropriate for the interpolation result.

- Additional time events are not possible in case of interpolation with Akima splines. The time coordinates are required to be strictly increasing. (Thus care must be taken when changing the interpolation smoothness if there are repeated sample points.)

The static function `isNearlyEqual` from source file `ModelicaStandardTables` is used to tell if two double-precision floating-point numbers are (nearly) equal with relative threshold `_EPSILON` (set to  $10^{-10}$  by default).

The calculation of the next time event  $t_E$  is performed in the interface function `ModelicaStandardTables_CombiTimeTable_nextTimeEvent`. For numerical reasons related to the floating-point arithmetic used, the current time  $t$  is incremented by a small fraction of the time table length, that is  $\_EPSILON \cdot (t_{max} - t_{min}) > 0$ . It is guaranteed that  $t_E$  is greater than the current time  $t$  and that no time events are missed if the distance between two consecutive time events is greater than this numerical increment.

If no future time event is found `\_nextTimeEvent` returns `DBL_MAX`. Hence, care should be taken by a Modelica simulation tool to avoid floating-point overruns during the event handling.

For debugging purposes, time events can be traced (by means of `ModelicaFormatMessage`) by defining `DEBUG_TIME_EVENTS` where each message line corresponds to one time event. For instance, linear interpolation and extrapolation of the  $4 \times 2$  example time table  $[0.25, 30; 0.5, 40; 0.5, 10; 0.75, 30]$  results in three time events (Fig. 1).

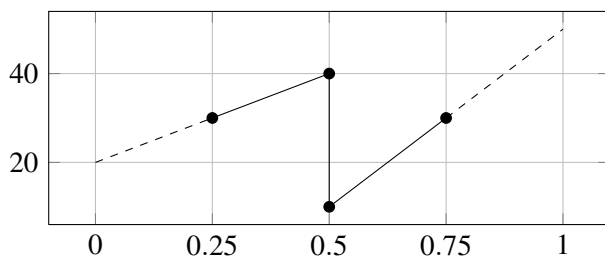


Figure 1: Linear interpolation of time table  $[0.25, 30; 0.5, 40; 0.5, 10; 0.75, 30]$  results in three time events.

The four messages (including the initial event) are

```
At time 0.00000: 1. time event at 0.25000
At time 0.25000: 2. time event at 0.50000
At time 0.50000: 3. time event at 0.75000
No more time events for time > 0.75000
```

In order to return the correct function values at the time events (i.e. during the event iterations), the interface functions `\_getValue` and `\_getDerValue` require not only the next event time but also its pre-value as input arguments.

## 2.2 Periodic extrapolation

For tables with periodic extrapolation, the numerically stable detection of periodic time events is rather tricky.

A first implementation was discarded as it turned out that the periodic time events were not reliably detected in all cases. The main reason is due to the IEEE 754 binary floating-point arithmetic where  $t$  and  $t - n \cdot T$  cannot be used simultaneously to detect the exact location of an event interval or periodic time event. Here,  $t$  denotes the floating-point number (FPN) of the current time and  $T = t_{max} - t_{min}$  the FPN of the period of the table. Variable  $n$  is a signed integer and denotes the multiple of the period. Furthermore, let  $t_E$  be the FPN of the event time (to be detected) and  $t_i$  the FPN of the corresponding event interval boundary time from the table. There are FPNs such that the inequality  $t \geq t_E = t_i + n \cdot T$  is true, i.e. indicates a time event, but where a simple rearrangement of the inequality to the form  $t - n \cdot T \geq t_i$  does not hold. This illustrates the fact that floating-point operations cannot be used to exactly evaluate floating-point comparisons, and therefore cannot be used to reliably detect periodic time events.

The final implementation is based on (nonnegative) integers

- `nEventsPerPeriod`, the number of time events per period and
- `eventInterval`, the (discrete) event interval marker.

During the very first call of function `\_nextTimeEvent`, the number of time events per period is determined from the time coordinates of the table array. There is always a time event per period at the

table boundary, thus  $n\text{EventsPerPeriod} \geq 1$ . Furthermore, the start and end indices of each of the so-called *event intervals* are determined and stored in the integer array *intervals*.

As an example, the  $5 \times 2$  time table  $[0, 10; 0.25, 30; 0.5, 40; 0.5, 10; 0.75, 30]$  is considered.

- There are two time events per period in case of linear interpolation. The indices of the event intervals are  $[0, 2]$  and  $[3, 4]$ , i.e. the first interval ranges from time 0 to time 0.5 and the second interval from 0.5 to 0.75 (Fig. 2).

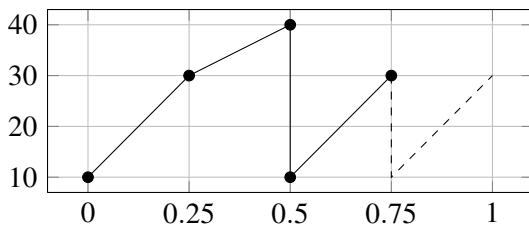


Figure 2: Two time events per period

- There are three time events per period in case of interpolation by constant segments. The indices of the event intervals are  $[0, 1]$ ,  $[1, 2]$  and  $[3, 4]$ . Repeated sample points are ignored since their ordinate values can never be taken by the interpolating function (Fig. 3)

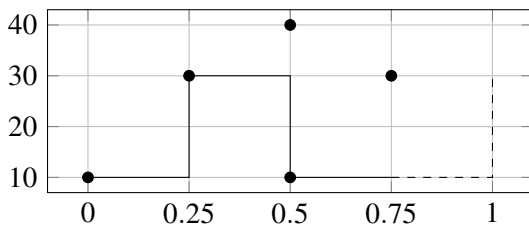


Figure 3: Three time events per period

The event interval marker `eventInterval` uses one-based indexing. It is properly initialized once in the very first call of function `_nextTimeEvent` since interpolation does not need to start in the first event interval by default. Along with both integer variables and the event intervals array, the initial offset time  $t\text{Offset} = n \cdot T$  is determined and stored.

Each subsequent call of `_nextTimeEvent` now increments the event interval marker by one and resets it to 1 once it overruns, i.e. gets greater than the number of time events per period. This can be derived from the fact that the input variable (time) is usually increasing (w.r.t. time). Then it is known that there is exactly one time event per event interval. There is

an event interval correction implemented in functions `_getValue` and `_getDerValue` that sets the time to be interpolated to either the left or right event interval value in case of floating-point inaccuracies. This implementation guarantees that no time events are missed and that the correct function values of the interpolating function at the event interval boundaries are returned.

## 3 Interpolation by Akima splines

Hiroshi Akima's original articles [1, 2] were used for the implementation of the univariate and bivariate interpolation by Akima splines. His reference implementation (in FORTRAN 77) was not used. Furthermore, the FORTRAN 90 code of SOSIE [3] from Laurent Brodeau was studied for an efficient calculation of the coefficients for bivariate splines.

### 3.1 Spline coefficients

There are different possibilities to calculate and store the polynomial spline coefficients of the interpolating function.

1. Pre-calculate all coefficients during the initialization and store them within the external table object.
2. Calculate the coefficients whenever needed (and possibly store only the last calculated set of coefficients if used for evaluating the partial derivatives in the same simulation time step).
3. Allocate enough memory during initialization to store all coefficients, calculate them whenever needed and store them whenever calculated. The advantage is that for very large tables where only a few different data parts are accessed there will be no superfluous calculation at all and initialization time is short. The disadvantages are the high memory usage and the unpredictable execution time of a simulation time step (as it is never known if the calculated coefficients are already available).

This implementation of the Modelica Standard Tables uses the first option where all required coefficients of the entire table array are calculated during initialization in function `spline1DInit` (for univariate Akima splines) and `spline2DInit` (for bivariate Akima splines).

### 3.2 Differentiability

The current univariate Akima spline interpolation always uses non-periodic table boundary conditions which may lead to a discontinuous interpolating function or derivative at the table boundaries for periodic extrapolation with the `CombiTimeTable` (Fig. 4).

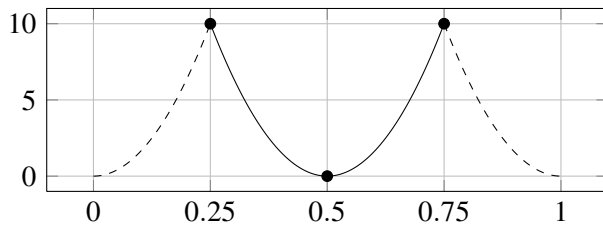


Figure 4: Periodic extrapolation of time table  $[0.25, 10; 0.5, 0; 0.75, 10]$  by Akima splines results in an interpolating function that is not (continuously) differentiable.

In all other cases it is guaranteed that the univariate or bivariate Akima spline interpolating function is continuously differentiable everywhere, especially at the table boundaries (Fig. 5).

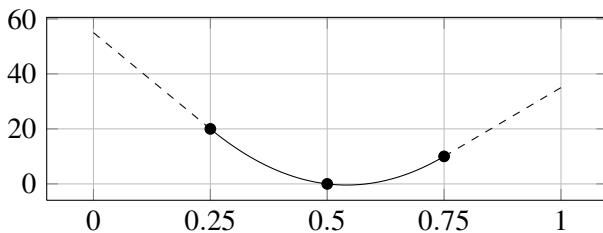


Figure 5: The boundary slopes of the Akima splines are used to linearly extrapolate the one-dimensional table  $[0.25, 20; 0.5, 0; 0.75, 10]$ .

## 4 Array memory optimizations

Advanced array memory optimization features are implemented and explained below.

### 4.1 Shared table arrays

If multiple table objects refer to the same table array of the same file, this table array is read and stored in memory multiple times since external objects are mutually independent by default. In order to avoid superfluous file input access and to decrease the utilized memory there is a C++ implementation of a global table array management on top of the C implementation,

guarded by predefined macro `__cplusplus`. The C or C++ compilation can be toggled by compiler flag `-x c` and `-x c++` for GCC or flag `/TC` and `/TP` for Microsoft Visual C++. In the case of a C++ compilation an additional static variable of type `TableShareMap` (using the `std::map` container from the STL) with reference counting is introduced. Write access of this global variable `tableShare` (e.g. insertion or erasure of tables) is thread-safe, i.e. guarded by a critical section (on Windows platforms) or `pthread_mutex` (on Linux platforms) and implemented by `struct CriticalSectionHandler`.

A table array update is usually not required and therefore not implemented. Shared arrays of spline coefficients are also not implemented, i.e. each external table object always calculates and locally stores the spline coefficients it requires.

### 4.2 Shallow copy of table arrays

This optimization is only relevant if the table array is defined within the simulation model, i.e. if it is known at compile-time and not read from the file at simulation run-time. The complete table array memory that is passed from the simulation model to the constructor (the `_init` function) of the external table object is allocated and copied (“deep copy”). This is the safe default case as nothing can be assumed about the lifetime of the table array of the simulation model. Thus the table memory is (temporarily) held twice: locally within the external table object and at the outside model. If the outside table array is known to be constant and to have a longer simulation lifetime than the external table object, the deep array copy can be avoided by defining `NO_TABLE_COPY`. In this case, the outside table array memory is used within the external table object (“shallow copy” of the passed array pointer).

## 5 Conclusions

An open source implementation of the table blocks

- `Modelica.Blocks.Sources.CombiTimeTable`
- `Modelica.Blocks.Tables.CombiTable1D`
- `Modelica.Blocks.Tables.CombiTable1Ds`
- `Modelica.Blocks.Tables.CombiTable2D`



is available with the MSL 3.2.1 as of August 2013. The Modelica code is provided under Modelica License 2. The C/C++ code of files `ModelicaStandardTables` and `ModelicaMatIO` [4] is provided under the new BSD license. As a result, all parts of the MSL are now available in a free implementation.

Additionally it should be mentioned that this implementation added new features to the table blocks.

- The new option `ConstantSegments` was added for the `Smoothness` parameter.
- The new option `NoExtrapolation` was added for the `Extrapolation` parameter.
- The table outputs can be differentiated once (with exception of the potentially discontinuous `ConstantSegments`).
- All MATLAB MAT-File formats are supported by an adapted library `libmatio` [4] (provided by `ModelicaMatIO` header and source file). Whereas MAT-File formats v4 and v6 are supported without additional dependencies by `libmatio`, the v7 format requires the `zlib` [5] library and compilation with preprocessor macro `HAVE_ZLIB=1` and the v7.3 format requires the `hdf5` [6] and `gzip` [7] libraries and compilation with preprocessor macro `HAVE_HDF5=1`.
- The support of tables provided as static C arrays in the user header file `usertab.h` was revised. This is relevant for real-time systems without a file system and where the table data is known at compile-time.

Last but not least, 120 test models in `Modelica-Test.Tables` with reference results have been created.

## 6 Acknowledgement

The presented work was paid for by the Modelica Association.

Grateful acknowledgements go to

- Martin Otter for the first implementation (from 1997 till 2001) of the table interpolation blocks and for constructive discussions on the improvement of the new implementation,
- Hans Olsson and Martin Sjölund for valuable advice on the improvement of the implementation,

- Christopher C. Hulbert for sharing a MAT-File C library and providing a patch of `Mat_VarReadDataLinear` [4],
- David Zaslavsky for initiating an `interp2d` library compatible with the GNU Scientific Library (GSL) [8],
- John C. Beatty for sharing a simple utility to concatenate multiple C source files into a single C source file [9] that was used to merge header and source files of [4] to single files `ModelicaMatIO.h` and `ModelicaMatIO.c`,
- Laurent Brodeau for sharing a bivariate Akima algorithm (in FORTRAN 90) [3].

## References

- [1] Hiroshi Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM*, 17(4):589–602, October 1970.
- [2] Hiroshi Akima. A method of bivariate interpolation and smooth surface fitting based on local procedures. *Commun. ACM*, 17(1):18–20, January 1974.
- [3] Laurent Brodeau. SOSIE is Only a Surface Interpolation Environment. <http://sosie.sourceforge.net>.
- [4] Christopher C. Hulbert. MAT File I/O library version 1.5.2. <http://matio.sourceforge.net>.
- [5] Jean-Loup Gailly and Mark Adler. A general purpose compression library version 1.2.8. <http://zlib.net>.
- [6] The HDF Group. Hierarchical data format version 5. <http://www.hdfgroup.org/HDF5>.
- [7] The HDF Group. Science data lossless compression library version 2.1. [http://www.hdfgroup.org/doc\\_resource/SZIP](http://www.hdfgroup.org/doc_resource/SZIP).
- [8] David Zaslavsky. A 2D interpolation library compatible with GSL. <https://github.com/diazona/interp2d>.
- [9] John C. Beatty. A very simple inclusion processor. <https://www.student.cs.uwaterloo.ca/~cs241/cLibs/mergeCsource>.



# The DLR Visualization Library

## Recent development and applications

Matthias Hellerer Tobias Bellmann Florian Schlegel  
German Aerospace Center (DLR), Institute of System Dynamics and Control  
82234 Wessling, Germany

### Abstract

Modelica models are mathematical descriptions and therefore their simulation output typically shows numerical variable trajectories. While universal for all kinds of simulations, this representation is oftentimes difficult to understand. In the field of multi-body simulations, 3D visualizations present a way of displaying vast amounts of numerical data in an intuitive way which is instantly understandable, even by people without specialized knowledge. For integration of visualization in Modelica multi-body simulations, the "DLR Visualization Library" has been developed. This paper presents the newest additions to the library and shows their application in several DLR projects.

*Keywords: Visualization, Simulation, DLR Visualization Library, Modelica, interactive Simulation*

## 1 Introduction

The visualization of simulation data is an important element of advanced simulations. With the increasing complexity of modern multi-body simulations, concerning for example flexible bodies, thermal dissipation, or contacts, the demands for a realistic, real-time capable visualization of the simulation also rise. Using Modelica as modeling language allows the user to pack model functionality in reusable sub-models, e.g. as a replaceable block for a suspension, an engine, etc. This also enables the integration of visualization definitions into the single sub-models, eliminating the need for an additional visualization definition in a separate program.

In 2009, the "DLR Visualization Library" for Modelica has been introduced under the now deprecated name "DLR External Devices Library" [1]. The library features a variety of visualization blocks to be used directly as visualizers within Modelica. Visualization elements like configurable rigid bodies (e.g.

sphere, box, gearwheel) or rigid and flexible bodies generated from CAD files can be displayed by connecting their respective Visualizer block to the corresponding frame in the multi-body model. Figure 1 shows an example scene created in Modelica and its visualization. Using the C-interface of Modelica, the visualization information is transmitted to the external visualization software which is started automatically

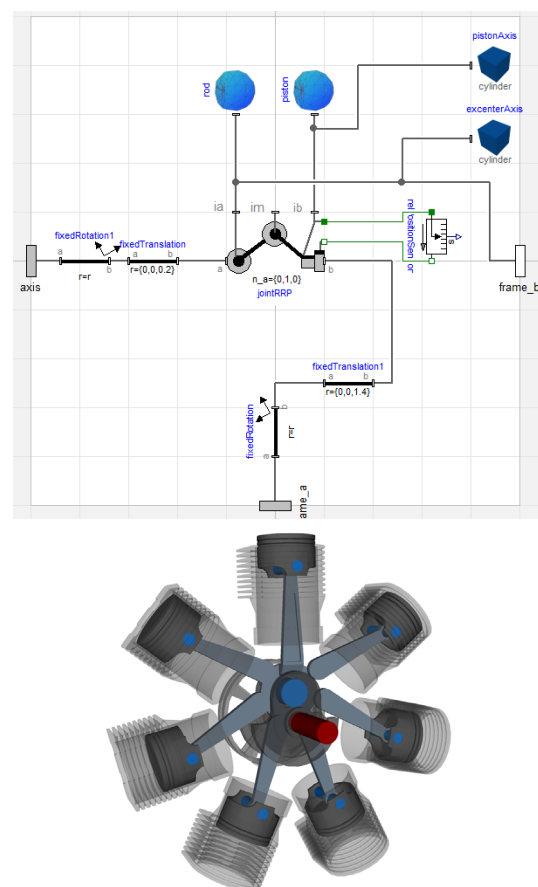


Figure 1: Top: Modelica model for a single cylinder, showing the integration of model and visualization. Bottom: The corresponding 3D visualization of a radial engine.

with the simulation and allows the user to replay and observe the simulation run in real-time and export it as video. Furthermore, as a result of the latest development, the library features a feedback channel, allowing the visualization to influence the simulation. This makes it possible to interact with the simulation using a graphical user interface (GUIs / HUDs), defined in the Modelica model, and to evaluate collisions between visualization elements, this is for example required in contact force calculations.

The following sections shall introduce the newly added features to the "DLR Visualization Library" and present some application examples of its use in ongoing research topics at DLR.

## 2 State of the art

The first attempts to create 3D Visualizations with Modelica was in 2000, when Engelson [2] started a discussion about techniques for the integration of visualization information with Modelica code. In his paper he presents a set of annotations for 3D graphic primitives and standardized simple geometries. Visualization would then have to be implemented by IDE tool vendors. Even though it brought up the idea of creating visualizations from Modelica models, the presented techniques were never implemented on a larger scale.

Yet, the idea of creating 3D visualizations from Modelica simulations persisted and in 2003 Otter et al. [3] revived the "MultiBody Library" of the Modelica standard library. This change introduced a new sub library called "Visualizers", containing shape objects for simple 3D forms as well as more complex objects from files. These are Modelica blocks and as such may easily be integrated in existing models or may be used in the creation of submodels which contain both simulation and visualization information. The visualization itself is redirected to the ModelicaServices library, responsible for all vendor specific implementations. So the Modelica standard library provides a standardized visualization description, to be implemented by Modelica IDE vendors. This technique is now part of all major simulation environments.

In 2008 Hoeft et al. [4] revisited the ideas of Engelson, this time integrating the powerful X3D standard in Modelica annotations. X3D, short for Extensible 3D Graphics, is an open international standard, developed for web applications. It is a representation of a 3D environment with XML. They present new annotations with X3D code and show an implementation of

the technique in the MOSILAB simulator.

Furthermore the Modelica3D library was presented by Höger et al. at the 2012 Modelica Conference [5]. It implements a scene graph in Modelica and uses the C-interface to connect the simulation front-end with the visualization back-end via interprocess communication. Besides their Modelica library, two different back-ends are presented. One based on the OpenSceneGraph 3D library and one based on the Blender 3D graphics software.

Unsatisfied with the existing visualizers, provided by "MultiBody Library", we implemented the "DLR Visualization Library". It is based on the ideas of Engelson and Otter, but tries to take those to the next level with more complex 3D environments, visualizations of mass and power flow, flexible deformation of objects and many other features in a high fidelity rendering.

## 3 New Functions

### 3.1 3D-Elements

#### 3.1.1 Dynamic textures

Given a 2D Matrix of 3D points, the flexible surface element may be used to display arbitrary shapes, with the ability to deform during the simulation. More details may be found in [1].

A new addition to flexible surfaces is the ability to display videos as textures, both local files as well as video streams from a network, as shown in figure 2. For local files the video is synced with the simulation time stamp. For network streams this is not possible and the surface will always display the last received image. The supported network protocols and the URL definitions for opening streams are inherited from the underlying FFmpeg library and are described in [18] in detail. Most commonly used protocols are thereby supported, such as: RTP, FTP, MMS, HTTP and HLS.



Figure 2: Full HD MPEG Video image from a presentation of the DLR Robotic Motion Simulator [9], rendered onto a flexible surface.

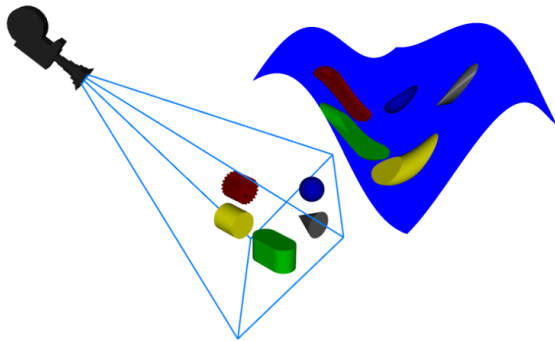


Figure 3: On the left a simple 3D scene; on the right this part is rendered on a flexible surface using a virtual camera, symbolized on the upper left side. All parts are in the same scene.

Besides Videos, images from virtual cameras can be rendered onto a flexible surface. For this purpose a standard camera from the "DLR Visualization Library" is placed in the scene and its output mode is set to "render to texture". The camera output can then be selected as texture for the flexible surface. The result is a camera image, drawn on the surface in the same way it would otherwise be drawn on screen. An example of this is illustrated in figure 3. The whole image shows one scene. On the left side a few objects are arranged and on the right side a flexible surface is depicted which shows the left part of the scene again, as seen from the virtual camera above the arrangement, rendered as a texture on a flexible surface.

### 3.1.2 Feedback - HUD

In [1] all communication was strictly from the simulation to the visualization. The following, new elements abandon this concept. They not only send data from the simulation to the visualization but also receive data, which may then be used to influence the simulation.

The first interactive objects presented here are interactive HUD elements. The base class for all of these elements is the button class. This defines an invisible HUD object in the visualization and with outputs in the simulation which are dependent on user input on the visualization. By combining the button object with HUD elements for the representation and Modelica logic for reactions, typical user interfaces, like buttons or sliders, can be created and used as interactive input for simulations. The button base class is capable of reacting to mouse-over events while the mouse cursor hovers over them, mouse clicks and dragging of the element by moving the mouse while a button

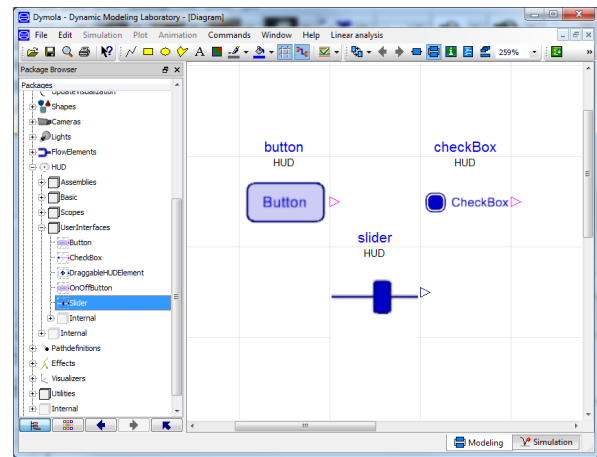
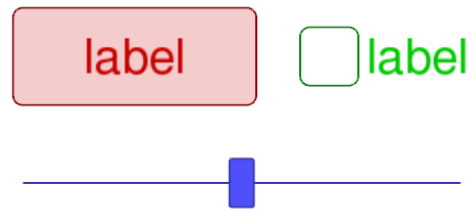


Figure 4: At the top part simple HUD elements are presented and below that their representation in the Modelica graphical designer.

is clicked. The "DLR Visualization Library" contains with a selection of predefined GUI elements, using the described button base class. The following HUD elements are depicted in figure 4:

- Buttons are by default visualized as simple squares with a label. They have a Boolean output value which is "true" for as long as the user presses it and "false" otherwise.
- Check boxes have a different look but actually behave very similar to buttons, except the output value behaves like a flip-flop. Permanently changing its value with every click.
- Sliders for simple adjustment of continuous values

### 3.1.3 Feedback - Collision detection

The description of complex 3D geometries usually uses geometric meshes. Reading the according data from files, interpreting it and running collision detection algorithms from Modelica is complicated. Physical simulations, especially in the area of multi body simulations often require to measure the distance between objects or contacts between objects. The required data, especially the interpretation and arrange-

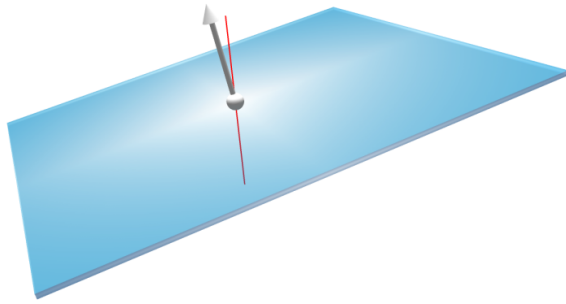


Figure 5: Collision detection with visualization of the contact. The red line represents the collision detector, the sphere shows the contact point and the arrow displays the surface normal.

ment of 3D mesh data, is already present in the visualization. To make this data accessible for the simulation the collision detector object is introduced. The collision detector is a line in the 3D scene with a defined start and end point. If the line intersects another element in the visualization it produces an output for the simulation. The output data includes a Boolean value, indicating whether or not a collision occurred, the distance of the first collision, measured from the start point and the surface normal vector of the colliding object at the position of the collision. The collision detection may also be visualized as shown in figure 5. The simulation and the visualization are running independently at different frequencies. In our example (see section 4.3), the Modelica simulation of the contact forces runs at a rate of  $1kHz$ , but the visualization calculating the collisions is bound to the frame rate of the graphics card and is as such dependent on the complexity of the scene and the hardware. By subsampling the communication calls from the simulation to the visualization, a minimum frame rate of about  $30 - 100Hz$  for the visualization is defined. If the visualization fails to achieve this frame rate, the simulation is slowed down below real-time speed as it has to wait for the answer of the visualization system providing the collision data. Because the simulation requires the collision data for each integrator step, a sample and hold interpolation is performed on the collision data, which saves the last received collision data until new ones are available. A high-level overview of this principle is depicted in figure 6.

### 3.1.4 Weather effects

To enhance the realistic impression of the simulation, the "DLR Visualization Library" provides a selection

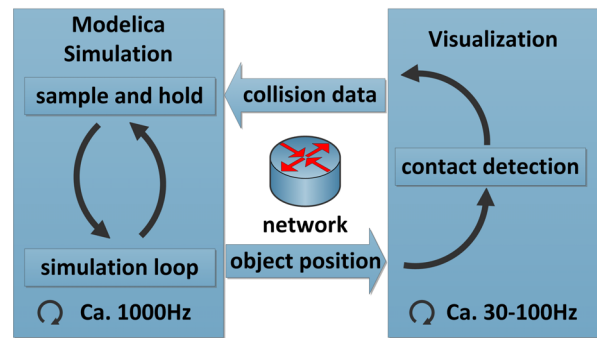


Figure 6: High level view of the collision detection system

of the most common weather effects for the simulation. These include: Rain, snow and fog. An example of the rain effect is shown in figure 7. For precipitation both the intensity of the effect as well as the wind direction and speed can be set. Wind can be used to simulate the effects of wind on the direction of precipitation. Fog is parameterized with three values: a starting distance for the fog effect and an end distance, where the view is completely blocked by the fog. Also the color of the fog can be adjusted. The applicability of those effects is not necessarily limited to the simulation of weather but could for example be used in under water simulation to show the effects of murky water.



Figure 7: An exemplary scene with rain effect

### 3.1.5 Paths

Visualizations often require complex motions of cameras and objects through space. This is type of predefined movement is seldom used in multi body simulations and therefore it is complex to define using the Modelica standard library. To simplify the creation of visualizations, new path definitions are provided. Paths are a multi-body connector frame with a time dependent output. Four different kinds of paths are

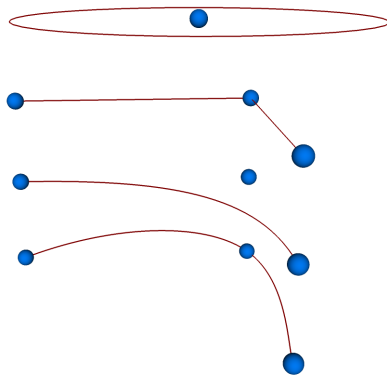


Figure 8: Top to bottom: circular, linear, Bezier, cubic spline; spheres: control points.

available as shown in figure 8: The first type shown here is a circular movement. The circle is defined by a center point with orientation and the radius. The simplest paths are point to point connection. A list of points is supplied and the resulting path is a linear interpolation between those. For smoother paths, Bezier curves can be used. The path is only guaranteed to pass through the first and the last point the remaining points only "stretch" the path towards them. The fourth kind of paths is based on cubic spline interpolation. Like Bezier curves these provide very smooth paths. In contrast those the line is always guaranteed to pass through all points. This often simplifies the definition of paths but might lead to overshooting in certain constellations.

### 3.1.6 Trace shape

Following the 3D movement of multiple objects at once can be hard. To display this motion, a trace-shape can be attached to any object. The trace-shape object then generates a line trail as the object it is attached to moves through space, thereby intuitively representing the objects trajectory. For the illustration of multiple trails, the line thickness, color and length can be adjusted by the user as needed. An example can be seen in figure 9.

### 3.1.7 Sky-Box

"Empty" space, areas on screen in which no object is defined, are by default filled with a solid color. This is suitable for small animations or abstract illustrations but for many applications a more realistic representation is required. For example in outdoor scenarios those areas should show the horizon and the sky. For this purpose the sky box is introduced. The sky box

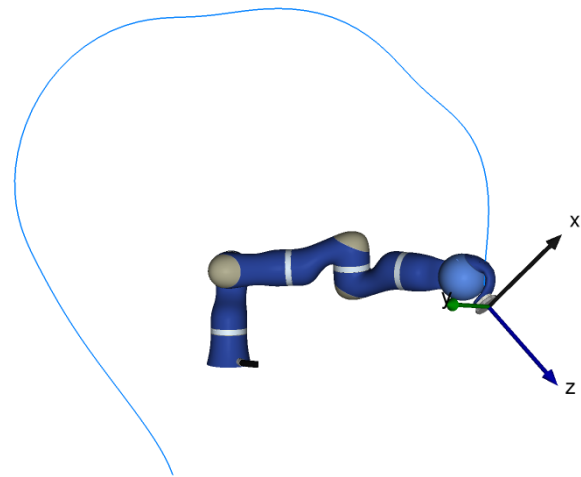


Figure 9: Movement of a robots tool tip visualized using the trace shape.

is always depicted as infinitely far away and is composed of two layers. The front layer is used for displaying some sort of horizon. An example would be distant hills or a tree line. The user has to supply six images (It is designed as a cube with the view point in the middle. One image per side). Behind this layer, visible through transparent areas in the images, the sky is drawn. The user has to provide a date, the time and a position on earth in form of longitude and latitude. The OsgEphemeris library [13] then uses this information to calculate the correct position of the sun and the moon during the day and an astronomically correct star field during the night time. The result is a realistic background for the simulation.

### 3.1.8 Particle System

The particle system is used for displaying objects, consisting of many sub-objects and cannot be modeled as traditional meshes. Examples for these kinds of objects include streams of water, fire, smoke and dispersed dust. Those objects are visualized using a large number of small objects, showing a simple image, called particles. Those particles are send out randomly from an emitter, follow a certain path while potentially turning, changing color and transparency, until the path reaches a certain length and the particle disappears. Using the right image, a large number of objects, and a specific movement, this raises for example the impression of water flowing from a pipe.

The Modelica blocks for modeling particles are divided by emitter type. The emitter is the area, from which the particles are shot. The three types available in the "DLR Visualization Library" are: Point emitter,

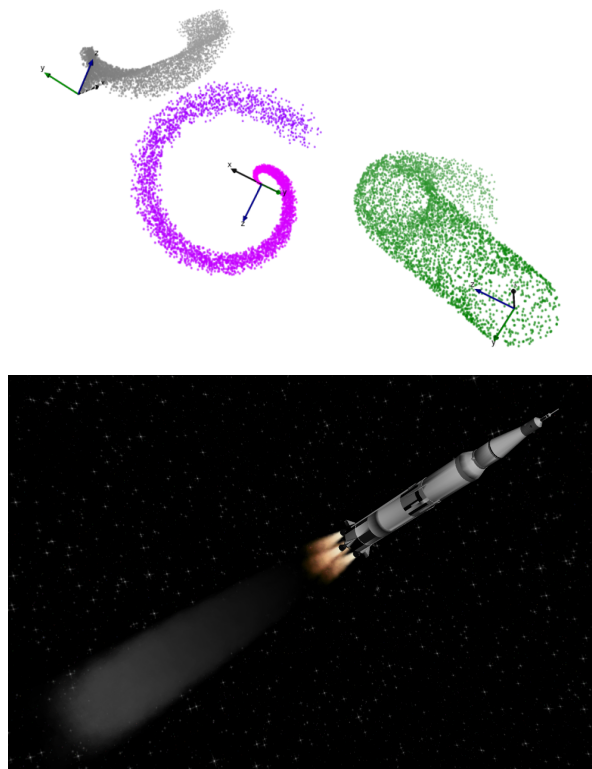


Figure 10: Top: Three types of particle emitters. From left to right line emitter (along  $z$ -axis), point emitter, and area emitter (circular shape in  $x - z$ -plane); Bottom: Particle effects used to display exhaust from a space rocket.

line emitter and area emitter. Examples for each type are depicted in figure 10. The particles are created randomly within certain bounds, such as mean number of particles per second, configurable for each emitter. Furthermore the particle may change its color, transparency, size over its life time. The final set of parameters then determines the path particles will follow.

### 3.1.9 Virtual Planet Builder and large scale terrain visualization

The visualization engine is based on the open source library OpenSceneGraph [12], allowing its use in conjunction with another OpenSceneGraph application, which is itself not part of the library: Virtual Planet Builder [14]. Using digital elevation data, like it is commonly produced in aerial surveys with special cameras, Virtual Planet Builder can produce 3D sceneries at scale of whole planets. This is accomplished by auto generating different levels of detail and by converting the scenery in a special file format. The planet surface is tiled into junks. When viewed from a faraway distance large areas are combined into

one large tile with very little detail. As the camera moves closer to a certain area the corresponding tile is split into smaller junks with more detail. This process is reiterated while the camera closes in until a maximum degree of detail is reached. For the viewer the switchover is not visible for the lower level detail at which a tile is rendered from afar, is not visible. Of special importance is the way the data is preprocessed and stored in a special format, allowing the renderer to load certain parts at different levels of detail as needed, very efficiently. This technique makes it possible to show extremely large areas while retaining a high level of detail. Figure 11 shows an earth model, created with Virtual Planet Builder from satellite images.

When rendering these planetary scale images in conjunction with small, close up objects, in setups like the satellite simulation in figure 11, graphical glitches appear. During the rendering process, each pixel is assigned a depth value to determine how objects overlap each other from the cameras perspective. The technical implementation of this uses a so called depth buffer or  $z$ -Buffer which saves each pixel distance from the camera (the depth or  $z$ -value). This is a, typically 24-bit on modern machines, fixed point value in the range  $[0, 1]$ , where 0 represents the near plane (minimum distance from camera) and 1 the far plane (maximum distance) of the view frustum. Therefore, with increasing distance between the near and far plane, the depth buffer resolution decreases. When the depth value of two points is so small that the depth buffer is unable to represent the difference graphical glitches become visible [15]. The minimum depth difference representable by the depth buffer  $\Delta z_{min}$  can be calculated with equation (1).  $z$  is the depth value,  $n$  is the near

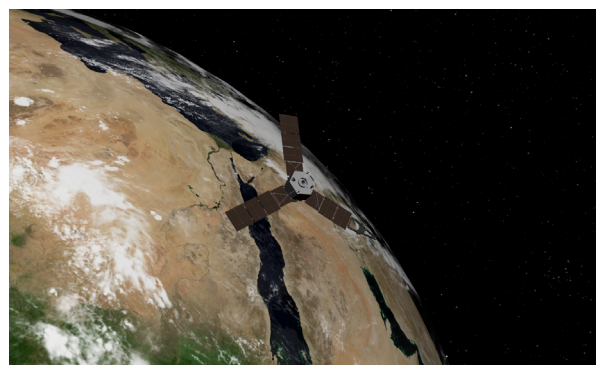


Figure 11: A model of planet earth, generated using virtual planet builder, in a satellite simulation; Logarithmic  $Z$ -Buffering allows for a detailed satellite model in the foreground and an earth model at real scale behind it.



plane distance and  $b$  is the depth buffers resolution in bit. The higher resolution at low distances is caused by perspective effects.

$$\Delta z_{min} \approx \frac{z^2}{2^b n - z} \quad (1)$$

Logarithmic depth buffers are a well known technique which seeks to attenuate this problem by applying a logarithm to the depth value before it is written to the depth buffer, thereby increasing the resolution for close objects, where small difference are more likely to be important at the expense of the resolution for distant objects where small differences are less likely to become visible. Equation (2) show the conversion from the original value  $z$  to the new value  $z_{log}$ . The addition parameter  $C$  is used to adjust whether close or distant objects are preferred.

$$z_{log} = \frac{2 \ln(Cz + 1)}{Cf + 1} - 1 \quad (2)$$

Using a logarithmic depth buffer alters the depth buffer resolution. It can now be calculated with equation (3). [16]

$$\Delta z_{log min} \approx \frac{\ln(Cf + 1)}{(2^b - 1) \frac{C}{Cz + 1}} \quad (3)$$

For example, the satellite visualization requires a render distance of  $1m$  to  $15000km$ . With a  $24bit$  depth buffer this results in a minimum depth separation of less than  $1mm$  at the satellites distance of  $100m$ , but between  $\approx 550m$  and  $\approx 1100km$  at the earths distance of  $300km - 13000km$ , causing visible problems with its rendering. Using logarithmic depth buffers with a  $C$  Value of  $0.001$  the minimum separation for the satellite also lies below  $1mm$ , yet for earth it is in the range of about  $\approx 20cm - 7.5m$  which is well below the visible range at this distance.

### 3.1.10 Oculus Rift Integration

The Oculus Rift is head-mounted virtual reality display, currently under development by Oculus VR. At the time of this paper, it is only available as a developer preview version with a finalized consumer product in development. The head-mounted system depicted in figure 12 includes a display with a resolution of  $1280 \times 800$ , two fish eye lenses stretching the image to a field of view of about  $90^\circ - 110^\circ$  and a three degree of freedom rotational acceleration sensor [17]. With this device it is possible to generate a fully immersive experience in a 3D environment, nearly filling the user's entire field of view and following his motion as he moves his head. The "DLR Visualization



Figure 12: The Oculus Rift head-mounted virtual reality device.

Library" supports the currently available Oculus Rift Development Kit as alternative display device for any kind of already integrated cameras. All setup steps and the camera orientation change as reaction to the users head movement are handled fully automatically.

### 3.1.11 HUDs

Head-Up-Displays are a two dimensional layer in front of the three dimensional scene. The possible applications for this kind of display range from overlaying logos, over the display of model state variables to complex interactive user interfaces. All user interfaces are composed of five base elements: The first element is text. Text can change dynamically and therefore be used to display model states or variables. Also typical text formatting methods such as different fonts, bold text and so on are supported. The second type of elements is a line drawn along a list of user pro-



Figure 13: Various HUD elements used to recreate a plane's cockpit instrumentation with artificial horizon, speed and altitude meters alongside a compass and engine status displays.

vided points. Next are free form faces, displaying a filled form based on polygon points provided by the user. Lastly, for more complex images, graphics files can be included. Of course the HUD elements presented here can be combined in the creation of new more complex elements. A small selection of such elements is included with "DLR Visualization Library". An example is a scope for the presentation of variable trajectories, a combination of lines, images and text, or the airplane cockpit instrumentation in figure 13. The last type is an interactive button base class which will be discussed in a later section.

Attention has to be paid to the arrangement of HUD elements for the size of the window they are drawn is variable even at runtime. Therefore it would be impractical to use absolute coordinates and instead relative coordinates are employed. For the adaptation to a changed display size the user can choose of the following four techniques:

1. The horizontal coordinate value spans over the horizontal display size and the vertical value is adjusted to its size in such a way that aspect ratio is preserved.
2. The same as 1 but for the vertical instead of the horizontal coordinate value.
3. The program automatically changes between technique 1 and 2, depending on the smaller side.
4. Relative coordinates range for both horizontal and vertical coordinates range over the whole display.

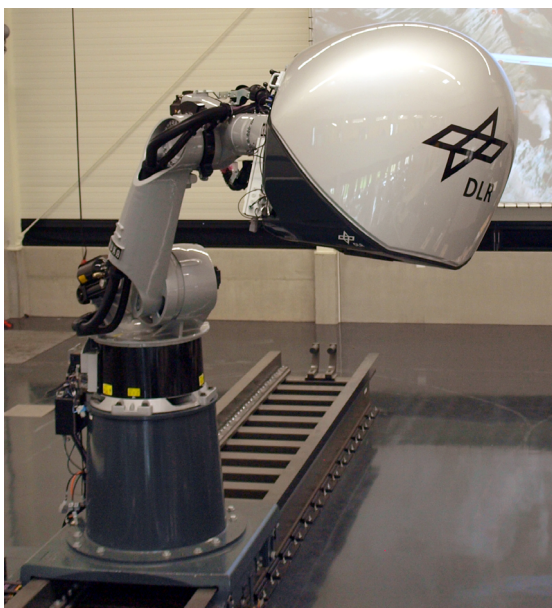


Figure 14: The "DLR Robotic Motion Simulator"

play size. The aspect ratio might be distorted depending on the display window setup.

## 4 Applications

This section exemplifies the real world application of the previously introduced visualization objects with help of their implementation in the "DLR Robotic Motion Simulator". An experimental motion simulation platform, currently under development at DLR. The system is depicted in figure 14.

### 4.1 ROboMObil GUI

The ROboMObil is a robotic car, developed at the DLR with four separately rotatable wheels, granting a new level of maneuverability. To make use of it, the specialized user interface in figure 15 was developed, using the previously described HUD elements. It is displayed on a touch screen device in the cockpit of the real prototype car

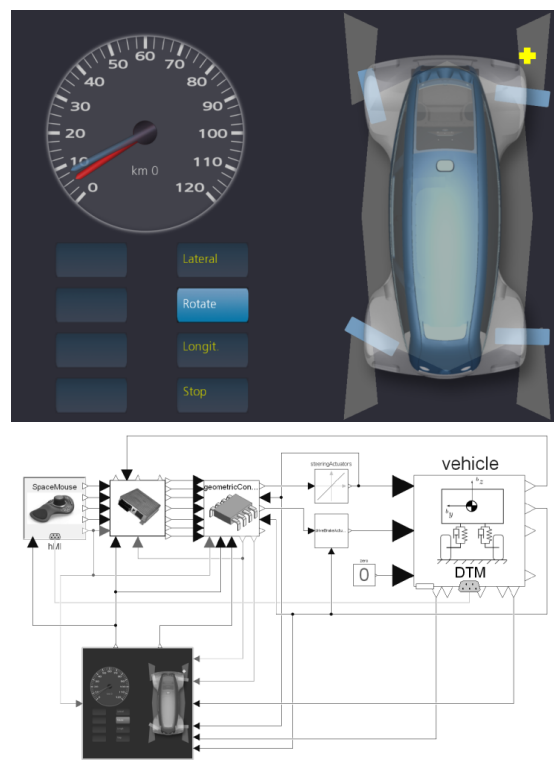


Figure 15: The ROboMObil GUI, as shown on the touch sensitive device of the ROboMObil cockpit, as well as the graphical Modelica designer. Left bottom: the user can choose between different operational modes; right: Enter additional options for the selected mode. In the rotational mode shown here, the user can select an instantaneous center of rotation

and in the simulators cockpit mockup. The concept is explained at full detail in [8]. The upper left part shows the current and the target speed. Below that the driving mode can be selected and the right part is selecting a rotation center point, required for certain driving modes. The second part of this image shows the full integration of the GUI with other parts of the simulation directly in the model description with Mod-  
elica.

## 4.2 Flexible trajectory planning

The design of a complex trajectory in 3D space, for a predefined movement of objects or cameras, can be a rather difficult and time consuming task, if it has to be accomplished by the direct manipulation of numeric parameters. Therefore the interactive Trajectory Designer has been developed to provide a convenient way of creating control point parameters for a B-spline interpolated movement of position and orientation. The trajectories velocity is interpolated with a two times derivable sinusoidal function.

In order to generate a smooth trajectory through accurately defined bases containing a position and orientation, we use quaternions for orientation and B-Splines for interpolation between these bases. A B-Spline curve  $T(x)$  consists of control points  $P_i, i \in 1 \dots n - p$  and B-Spline base functions  $N_{i,p,\tau}, i \in 1 \dots n - p - 2$ :

$$T(x) = \sum_{i=1}^{n-p} P_i N_{i,p,\tau}(x). \quad (4)$$

The control points  $P_i$  each contains seven elements; three for position and four for the orientation represented as quaternion. A base function  $N_{i,p,\tau}$  is defined as a polynomial piece with order  $p$  and knot vector  $\tau$ :

$$N_{i,0,\tau}(x) := \begin{cases} 1, & x \in [\tau_i, \tau_{i+1}] \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$N_{i,p,\tau}(x) = \frac{x - \tau_i}{\tau_{i+p} - \tau_i} N_{i,p-1,\tau}(x) + \frac{\tau_{i+p+1} - x}{\tau_{i+p+1} - \tau_{i+1}} N_{i+1,p-1,\tau}(x) \quad p > 0 \quad (6)$$

The knot vector  $\tau = [\tau_0, \dots, \tau_{n-1}]^T, n \geq 2p, \tau_i \leq \tau_{i+1}$  and  $\tau_i \leq \tau_{i+p}$  has to be chosen. The algorithms of our implementation set the first and last  $p$  knots equal. There are different approaches setting the remaining values, see [11] [10, p161]. For the trajectory designer we use a base functions with degree 3 in order to get a smooth trajectory with a continuity of the second derivative. The parameter  $x$  defines the position on the trajectory,  $T(x)$  returns the interpolated

value  $P(x)$  containing the three-dimensional position and an interpolated quaternion. The quaternion interpolation corresponds to a linear quaternion interpolation (LERP)[6]. Therefore the resulting  $L_2$ -Norm of the vector is less than 1 and has to be normalized resulting in a unit quaternion which leads to an varying but continuous rotational velocity. Furthermore the trajectory designer provides the functionality to set a velocity at each control point. The behavior of the velocity between the points is computed through sinusoidal functions:

$$v(\lambda) = (\lambda - \frac{1}{2}\pi \sin(2\pi\lambda))(v_{i+1} - v_i) + v_i; \quad (7)$$

with  $v_i$  the velocity of the left and  $v_{i+1}$  the velocity of the right control point within the interval between  $P_i$  and  $P_{i+1}$ . The second derivative of this sinusoidal function is continuous and zero at the left and right end providing a smooth transition at the control points. The integration of the velocity  $v$  leads to the current position on the trajectory  $x$ . Therefore,  $v_i \geq 0, i \in 1 \dots n - p$ .

The tool is operated by a small selection of keyboard commands and offers three operating modes to manipulate the position, the orientation and the associated velocity of the control points. In all modes control points can be removed or inserted. For verification of the resulting interpolated movement a live preview, adjusting to the manipulation of control points in real-time, is available. The preview shows a coordinate system traveling along the spline in the main scene and

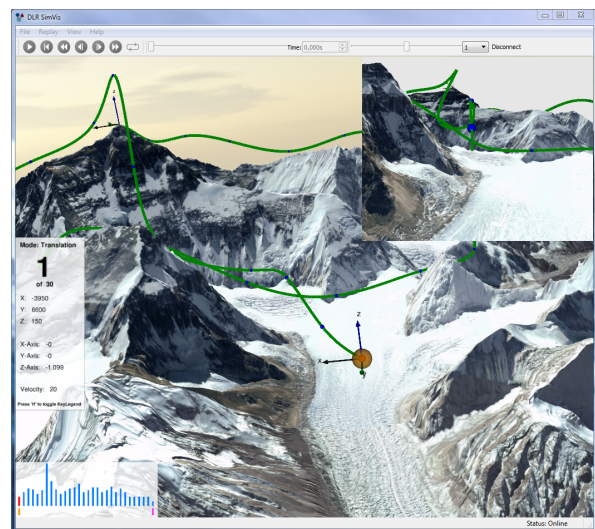


Figure 16: The trajectory designer tool showing a path (green line) above Mt. Everest. In the upper right corner a window previews the camera's trajectory while working on it.

a point of view rendering, adequate for the preview of camera movements, in a separate picture-in-picture preview area. The complete interface is depicted in figure 16. The coordinates, orientations and velocities are finally stored as vectors in a text file, which can be used to reload the control points for further manipulation or later playback.

### 4.3 Wheel ground contact

The simulation of cars requires the modeling of contacts between wheels and the ground, realized using the collision detector elements. An abstracted model is shown in figure 17. This contact model is then the basis for far more complex models, such as the car shown in figure 18. In this simulation the load on the wheels will shift when driving on uneven terrain and the suspension reacts accordingly. The ground plane in the simulation can be any 3D shape.

The wheel vertical force is calculated according to the following equation. The contact force  $f$  is the sum of a spring force  $s$  and a damping force  $d$ .

$$\vec{f} = \vec{s} + \vec{d} \tag{8}$$

The spring force is calculated using the penetration depth of the collision object and an other object  $p$ , as well as a spring constant provided by the user  $S$ , to push the wheel away from the other object, along the

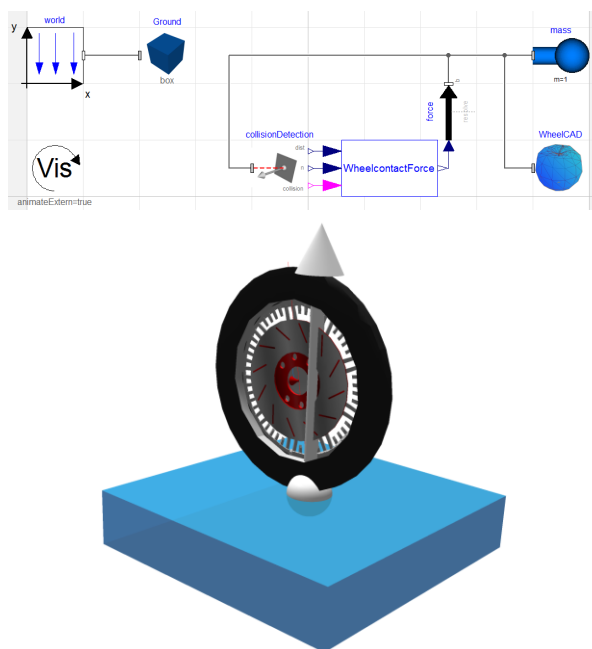


Figure 17: Wheel ground contact in the graphical Modelica designer and as 3D visualization.



Figure 18: A car driving on a plane. White arrows indicate surface normals for contact points.

objects surface normal at the point of contact.

$$\vec{s} = S \cdot p \cdot \vec{n} \tag{9}$$

Just using a spring force would result in a constantly bouncing wheel. To model energy dissipation, a damping force  $d$  is introduced. It is calculated using the collision objects speed  $\dot{r}$  in the direction of the surface normal  $n$ , a user provided damping constant  $D$  and the resulting force, just as the spring force, acts in the direction of the normal. Lastly the damping force should only be present during impact. Otherwise the wheel would act as if it was glued to the surface.

$$\vec{d} = D \cdot \min(0, \dot{r} \cdot \vec{n}) \cdot \vec{n} \tag{10}$$

This way of simulating object collisions does come with certain draw backs. First of all, it intrinsically requires the two colliding objects to interpenetrate. While problematic for rigid bodies, it is a reasonable approximation for flexible objects like the car tires in the presented example. When trying to minimize the interpenetration a further problem arises. The larger the spring constant  $s$ , the stiffer the simulation gets, requiring ever smaller simulation time steps. Otherwise the interpenetration from one step to the next can be so large, the resulting force from equation (9) gets unrealistically large, hurling the wheel away. The same problem can occur when the wheel is moving with high speed and the ground inclination changes. The sample-and-hold technique used in the communication, delays the point in time when the simulation is able to "see", the changed ground inclination. This is depicted in figure 19. The wheel on the left moves at high speed to the right but due to the slower visualization time steps, the ground inclination change is communicated to the simulation with delay, causing the wheel to penetrate the ground.

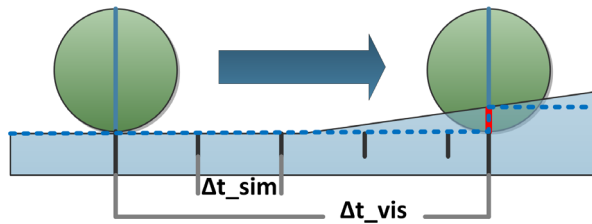


Figure 19: Wheel moving at high speed from left to right. Due to the fact that the simulation does run faster than the visualization, the inclination change is recognized to late. Dotted line: ground level as detected by the simulation; red line: error due to sample and hold technique

For small and fast moving objects, it is even possible for object collisions to be missed completely. The collision detection object only detects collisions with object surfaces so if an object moves so fast towards another, that the position "jump" between time steps is larger than the collision object length, the collision might get missed. This case is described by equation (11) where  $v_1$  and  $v_2$  the speed of the two objects and  $i_1$  and  $i_2$  are start and end point of the collision object.

$$\left\| \frac{v_1 - v_2}{\|v_1 - v_2\|} \cdot \frac{i_1 - i_2}{\|i_1 - i_2\|} \right\| \frac{1}{f} > \|(i_1 - i_2)\| \quad (11)$$

#### 4.4 Image warping

The capsule in figure 20 is part of a simulator project. At the back, besides the head of the pilot, are two projectors. The projection screen is the open capsule shell to the top right. The shell has a complex geometry, deforming the images projected onto it. In order to present a rectified image to the pilot a reverse deformation has to be applied to the image prior to projection. This preprocessing utilizes the render image to texture functionality on a flexible surface. Due to the fact that a flexible surface is used, the image can be warped as needed (see figure 20) and with the correct configuration, the final image appears restituted to the pilot.

#### 4.5 Manned vehicle simulation

The "DLR Robotic Motion Simulator" uses the capsule shown in figure 20 and an industrial robot to which the capsule is attached as shown in figure 14. The robot is then used to apply accelerations, in accordance with the simulation, on the pilot, thereby creating an immersive motion simulation. A detailed description of the "DLR Robotic Motion Simulator" can

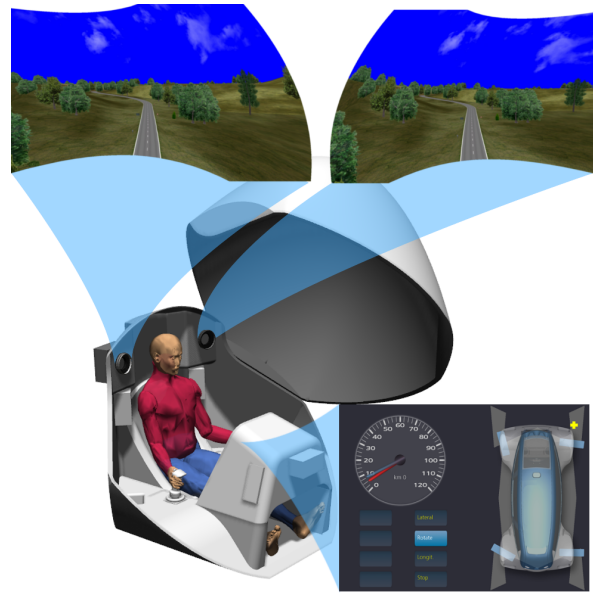


Figure 20: The piloting capsule of the "DLR Robotic Motion Simulator"; On top the stereo images after warping; The images are projected on the capsule and appear restituted; In front of the pilot is a touch sensitive display.

be found in [7]. In this application all of the previously shown applications are utilized. The pilots' main screen is restituted using the render to texture feature on a flexible surface; the console in front of the pilot shows an input GUI on touch screens, and the vehicle simulation uses collision detection objects for the wheel ground contact analysis.

## 5 Limitations

The presented library does have certain limitations in its current state, of which the following three are currently under investigation for improvements. The first one is the design of the collision detection system: it only allows for collisions with a line object, limiting its use to applications where the point of contact is predictable, such as the presented tire, where the contact point can be assumed to be in the direction of gravity, while arbitrary contact points, like the collision of a car with a pole, can not be modeled in a similar fashion. Also the underpinning architecture, as introduced in [1], only permits retroactive collision detection. It only detects interpenetration of the collision object with another object after it happened and without any possibility of detecting the exact time of contact. Any contact model relying on the collision data has to account for this. The second item for improve-

ment is the graphical fidelity. While the current system provides very high fidelity compared to other products for scientific visualization, it does not hold up when compared to state of the art graphics as seen in modern computer games. The visualization of simulation data might not require such graphics, yet in virtual reality applications the best graphics possible are desired to maximize the users level of immersion. For comparison between our solution and a modern computer games engine see figure 21. For better visibility a very simple scene was selected here. On the top our solution is shown, with shadows enabled. Below that, the same scene is displayed using the Unity3D engine [19], with deferred lighting, advanced soft shadows, field of view and screen space ambient occlusion among other effects. Even though it is very simple, the second scene looks more realistic. Finally, the visualization library is based on Modelica's multi-body library. In virtual reality applications a large number of visualization elements gets connected using frame connectors. Even for static and fixed compositions the number of resulting equations gets extraordinarily high. An empty scene, with only the multi-body world object and the visualization libraries update-Visualization object, requires 1073 equations. Each

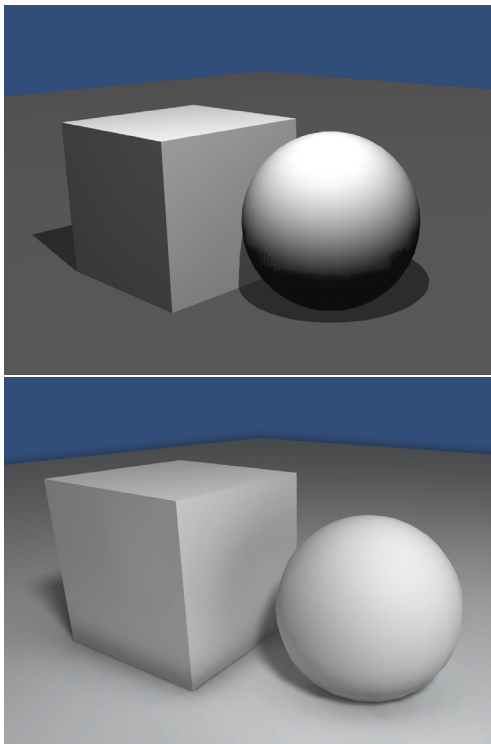


Figure 21: A simple scene for comparison between our current visualization on the top and below a modern computer games engine

additional ElementaryShape (e.g. a simple box) introduces 217 equations and each fixedRotation object used to arrange the objects in the scene further requires 102 equations. Clearly this modeling is too complicated and for complex simulations it can lead to performance problems. Since described problem is caused by Modelica's design of the multi-body library, we propose a simplification of the connector for the case that no masses are involved, when the multi-body library gets reevaluated in the future.

## 6 Conclusion

Visualization is an important, if not necessary, augmentation for a multitude of simulations. The "DLR Visualization Library" provides a sophisticated visualization framework for the Modelica modeling language. The paper presented the new additions to the library: videos and camera images rendered on flexible surfaces, advanced user interfaces, a collision detection system, weather effects, paths, the trace shape, a particle system, sky-boxes and integration with Virtual Planet Builder and support for virtual reality hardware. Furthermore, real-life applications for these new elements were presented as they are used in the development of the "DLR Robotic Motion Simulator".

In comparison with the other existing libraries, our implementation is not based on annotations and therefore does not rely on vendor specific annotations. Also it is not only possible to render all visualizers described in the standard MultiBody library but it also heavily augments its rather limited possibilities. In relation to other solutions, the "DLR Visualization Library" provides the richest feature set along side high fidelity results.

The visualization component is currently developed for Windows XP, 7, 8 and a Linux version is in Beta test with a installation package for Ubuntu 12.04 available. The library itself utilizes the Modelica C-interface and should therefore be compatible with a wide range of simulation environments but currently only Dymola has been tested extensively and is officially supported.

In conclusion, the library proves useful for gaining an intuitive understanding of multi-body simulations, the creation of presentable results and the creation of interactive virtual reality environments.

## References

- [1] Bellmann Tobias. Interactive Simulations and advanced Visualization with Modelica. Como, Italy: Proceedings of the 7th International Modelica Conference, Linköping University Electronic Press, 2009.
- [2] Engelson Vadim. 3D Graphics and Modelica - an integrated approach. Linköping, Sweden: Linköping Electronic Articles in Computer and Information Science, 2000.
- [3] Otter Martin, Elmqvist Hilding and Mattsson Sven Erik. The New Modelica Multi-Body Library. Linköping, Sweden: Proceedings of the 3rd International Modelica Conference, Linköping University Electronic Press, 2003.
- [4] Hoefft Thomas and Nyscht-Geusen Christoph. Design and validation of an annotation-concept for the representation of 3D-geometries in Modelica. Bielefeld, Germany: Proceedings of the 8th International Modelica Conference, Linköping University Electronic Press, 2008.
- [5] Höger Christoph, Mehlhase Alexandra, Nyscht-Geusen Christoph, Isakovic Karsten and Kubiak Rick. Munich, Germany: Proceedings of the 9th International Modelica Conference, Linköping University Electronic Press, 2012.
- [6] Erik B. Dam, Martin Koch and Martin Lillholm. Quaternions, interpolation and animation. Datalogisk Institut, Københavns Universitet, 1998.
- [7] Bellmann Tobias, Heindl Johann, Hellerer Matthias, Kuchar Richard, Sharma Karan and Hirzinger Gerd. The DLR Robot Motion Simulator Part I : Design and Setup. Shanghai, China: IEEE International Conference on Robotics and Automation, 2011.
- [8] Bünte Tilman, Brembeck Jonathan and Ho Lok Man. Human Machine Interface Concept for Interactive Motion Control of a Highly Maneuverable Robotic Vehicle. Baden-Baden, Germany: Speech, 2011 IEEE Intelligent Vehicles Symposium (IV), 2011.
- [9] DLR. DLR Robotic Motion Simulator - Driving Simulation.: Website/Video, 2013. <http://www.youtube.com/watch?v=QCGcWOWv8Qs>
- [10] Gerald Farin. Curves and Surfaces for CAGD - A Practical Guide. 5. edition. : Morgan Kaufmann, 2001
- [11] Gerhard Schillhuber. Geometrische Modellierung oszillationsarmer Trajektorien von Industrierobotern. Munich, Germany: Technische Universität München, 2002
- [12] Osfield Robert and others. OpenSceneGraph.: Website, 2013. <http://www.openscenegraph.org/>.
- [13] osgephemeris - An ephemeris model for use with OpenSceneGraph.: Website, 2013. <http://code.google.com/p/osgephemeris/>.
- [14] Osfield Robert and others. openscenegraph/VirtualPlanetBuilder - GitHub.: Website, 2013. <https://github.com/openscenegraph/VirtualPlanetBuilder>.
- [15] Khronos Group. The Depth Buffer. Beaverton, Oregon, USA: Website, 2013 <http://www.opengl.org/archives/resources/faq/technical/depthbuffer.htm>
- [16] Patrick Cozzi and Kevin Ring. 3D Engine Design for Virtual Globes. : CRC Press, 2011
- [17] Oculus VR, Inc. Oculus Rift - Virtual Reality Headset for 3D Gaming | Oculus VR. Irvine, California, USA: Website, 2013. <http://www.oculusvr.com/>.
- [18] FFmpeg team. FFmpeg Documentation.: Website, 2013. <http://ffmpeg.org/ffmpeg-protocols.html>.
- [19] Unity Technologies. Unity - Games Engine. San Francisco, California, USA: Website, 2013. <http://unity3d.com/>.





# Automated Modelica Package Generation of Parameterized Multibody Systems in CATIA

Daniel Baumgartner, Andreas Pfeiffer

German Aerospace Center (DLR), Institute of System Dynamics and Control  
82234 Wessling, Germany

Daniel.Baumgartner@dlr.de, Andreas.Pfeiffer@dlr.de

## Abstract

In early stages of the product development process computer-aided design (CAD) and multibody simulation (MBS) work concurrently to build a virtual mechanical system. While CAD handles the geometric design and space analysis, MBS leads to a deeper understanding of the dynamic behavior of the future system. The CAD system has to provide physical and geometrical data, such as mass, inertia and connecting frames in order to improve simulation results. Automation at this point helps to create consistent simulation and design models and shortens the amount of time needed to produce realistic simulation results. Based on Visual Basic for Applications (VBA) a method is implemented to automatically generate an isolated Modelica model package or a single Modelica model from CATIA assemblies or parts. The introduction of design controlling parameter variables in addition to the multibody data enables optimization loops between multibody simulation and the related CAD model. An example demonstrates the three main steps of the presented method, divided into model processing, package generation and parameterized package update. Furthermore, the update process is integrated in a manual parameter variation as well as an automated optimization routine to enable parametric design studies coupled with multibody simulation.

*Keywords: CATIA, Modelica, Dymola, Multibody Simulation, Parameterized Models, Package Generation, Optimization*

## 1 Introduction

Multibody simulation and computer-aided design are gaining significantly in importance during the virtual product development process. Typical tasks of multibody analysis include the computation of dynamic behavior, coupling forces or modal analysis. CAD systems on the contrary provide a three-

dimensional representation of virtual assemblies and subordinate parts. During the development process both engineering disciplines need to exchange data in the loop. Figure 1 clarifies the relations between the two domains. Physical and geometrical data is extracted from CAD and used as input for MBS. Dynamic simulation and optimization proposes changes in the design. Traditionally both domains have independent experts for the applied software. The generated data is shared manually between them over agreed-upon fixed interfaces. Automating this time consuming and error-prone repetitive procedure is a step toward the improvement of the product development process. The automation approach depends on the used design and simulation software. In the context of the presented work Modelica [1] is used as modeling language for multibody simulation and CATIA [2] as design software.

Existing methods for automated translation from CATIA to Modelica take different boundary constraints into account. The approach in [3] is based on a kinematic skeleton in Modelica. This Modelica file is imported in CATIA and the user maps CATIA data to the kinematic skeleton through a graphical user interface. Finally the multibody data is exported to the Modelica file. In this approach the design and the simulation work is divided and executed by different persons. The design process is the master. [4] presents a method for kinematic coupling of CATIA models with Modelica. The basic requirement for this method is a working kinematic structure in CATIA. The kinematic structure from the CATIA

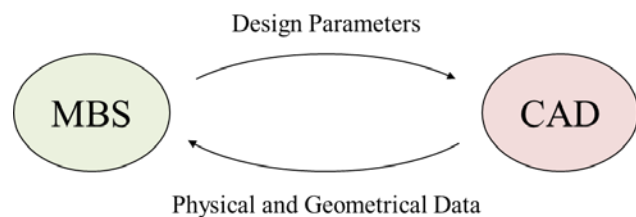


Figure 1: Relation between MBS and CAD

kinematics workbench is translated to Modelica Code with attention to kinematic loops and special treatment of redundant joints.

In some cases, however, this time-consuming setup of a kinematic structure in CATIA is not desired or not possible due to open kinematic chains. In terms of dynamic preinvestigation at early stages of the product development process, however, it can be helpful to focus on the simulation process as master in order to generate fast results. Therefore a workflow is presented here, which matches the aforementioned requirements. An isolated package or a model is generated from CATIA assemblies and parts to be used for simulation and optimization in a Modelica simulation environment, based on the Dymola framework. The mandatory model preparation work can be executed solitarily by a simulation expert with basic CATIA experience.

The upcoming sections of this paper are organized as follows. Section 2 gives a detailed overview of the used software and presents the classic manual approach of virtual development. The following section presents an automated translation approach divided into model processing, package generation, parameter handling and package updating. A bicycle rear suspension serves as example and demonstrates the usage of this method in Section 4. Section 5 summarizes the main conclusions together with an outlook to future work.

## 2 Multibody Simulation and Computer-Aided Design

This section presents the tools used in the context of the paper. Firstly, multibody simulation in Modelica with Dymola is described. The following subsection focusses on the design process in CATIA. Subsection 2.3 finally describes the classic workflow of product development between the presented domains.

### 2.1 Multibody Simulation in Modelica/Dymola

In general, multibody models consist of rigid bodies and joints. The bodies are modeled as a skeleton of fixed translations  $r_{0n}^0$  and rotations  $T_{0n}$  between connecting frames together with body mass and inertias representing the real products shape (Figure 2). Joints between the components allow certain degrees of freedom and determine the kinematic behavior of the mechanical system. The Modelica Standard Library provides algebraic and dynamic models for multibody systems [5]. These can be combined

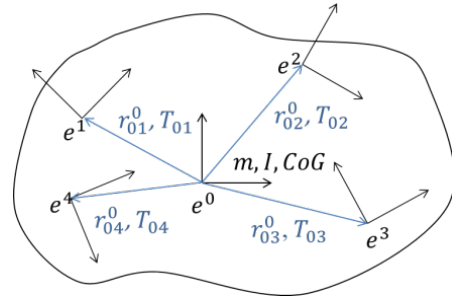


Figure 2: Geometric skeleton of a rigid body in a multibody simulation

graphically in Dymola to create multibody systems for dynamic model simulations. Based on the equilibrium of forces Differential Algebraic Equations (DAE) are formulated and solved. The outputs of the simulation experiments are motion, forces and modal behavior of the considered system. Model parameters can be adapted in optimization loops to enhance the system dynamics to match the desired behaviors.

### 2.2 Computer-Aided Design in CATIA

The design process in CATIA is mainly carried out by using the Part Design and the Assembly Design Workbench. The Part Design Workbench is used for modeling of single parts made from basic geometric elements such as points, lines and planes. These elements are combined to create two-dimensional sketches. Elementary operations transform these sketches to surfaces and volumes. Parameterizing variables, in the following referred to as design parameters, can be defined on top-level to create different representations of a part. In order to determine the part's mass and inertia, its density has to be defined through the assignment of materials.

The Assembly Design Workbench, on the contrary, is used to design mechanical systems from existing parts or sub products. Geometrical constraints, like coincidences and offsets, represent the locked degrees of freedom by the connecting joints between

Data Type	Parameters	Workflow
Physical Data	Mass, Inertia, Center of Gravity (CoG)	CAD to MBS
Geometrical Data	Connecting Frames, Points of Interest	CAD to MBS
Visualization	Shape representation	CAD to MBS
Design Parameters	Lengths, Angles	MBS to CAD

Table 1: Workflow between MBS and CAD

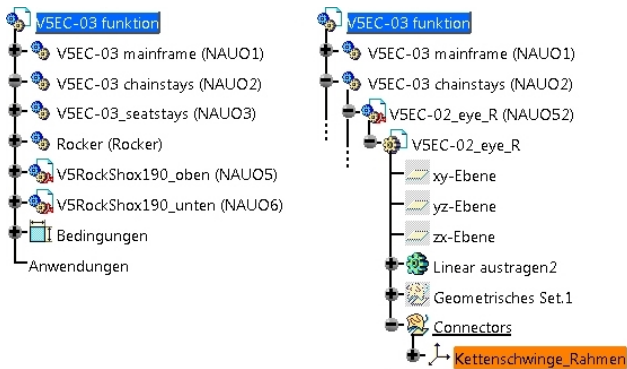


Figure 3: Top-level and connector level structure of an assembly in CATIA

the components. These relations allow manipulation according to the system's degrees of freedom for space and collision analysis. As previously mentioned multibody data is needed as input for the simulation work. There are different types of data: physical information, geometry, visualization and design parameters. Table 1 shows the workflow between MBS and CAD.

CATIA is organized in a hierarchic model structure. Figure 3 shows the top-level products and the nested structure on connector level. Each component of an assembly can consist of further products and parts on various levels. In terms of kinematic investigations it is sufficient to focus on the top-level of the product structure where the moving parts of a mechanism are located. Therefore the physical and geometrical data of the sub products is transformed to the first level of the assembly structure. Mass, inertia and center of gravity is computed for all sub products individually and merged to a single representative data set.

### 2.3 Classic Workflow

Traditionally the workflow can be described as follows. After the conceptual phase of the product design the kinematic structure is defined and initial modeling work in MBS and CAD is triggered. Important interfaces between both domains are fixed in order to establish consistent data exchange. Based on the early design model multibody data is extracted by the CAD expert and passed to the simulation expert. This data is used by the MBS expert to update the related simulation models. Accordingly, the MBS expert analyzes the dynamic model behavior and suggests changes to the design to match predefined objectives. In the next step these changes are applied to the design model and the multibody data is updated. These elemental tasks are repeated in the loop

until the dynamic behavior of the system meets the desired requirements.

Automating the repetitive data extraction from CAD to MBS helps to create consistent data and shortens the amount of time needed to produce realistic simulation results. As already mentioned, different approaches exist to improve the mutual work of design and simulation experts. At early stages of the product development process it can be more helpful to focus on the simulation process as master to rapidly generate simulation results. The presented approach in the next section handles the kinematic structure exclusively on the simulation side.

## 3 Automated Modelica Package Generation

The main contribution of the presented work is a method for an automated Modelica package generation with specific characteristics. The introduction of design parameters in addition to the multibody data allows parameter variations with MBS and CAD in the loop. Furthermore, the simulation expert is able to create simulation experiments based on an early design version while CAD work is still in progress.

The workflow of automated model generation is divided in three main steps. First, existing CAD data has to be processed in CATIA according to a defined structure. Second, after the preparation work an initial package or a model in Modelica is generated and used to setup simulation experiments. Finally, due to variation of design parameter values or manual changes to the CAD part or product the simulation model or package is updated in the loop. In what follows of this section, we will briefly analyze each one of these steps.

### 3.1 CATIA Model Preparation

Starting point is either raw CAD data from other software or existing CATIA parts or products. In terms of package generation the top-level structure of the root product has to be reorganized to match the kinematic structure of the considered mechanism. The generation of connector frames in Modelica is based on CATIA axis systems. Therefore specific geometrical sets with axis systems are created in selected parts of the product structure. These axis systems are either connecting points for joints or other points of interest such as measurement or load points used in simulation. These preparation steps are mandatory for the export of the multibody data. In order to increase usability for the optimization-based de-

sign process, additional design parameters can be defined on the part level. These parameters are either lengths or angles that are linked to certain geometrical constraints of the part. The design parameters change the parts appearance and hence it's mass, inertia and geometrical information. Design parameters need special treatment during the update process as described in the next subsection. The output of the model preparation work can be summarized as follows:

- Top-level product structure equal to kinematic structure,
- Assignment of appropriate materials (densities) to every part of the product structure,
- Connector frames as axis systems in geometrical sets on part level,
- Optional design parameters linked to constraints of the design model on part level.

### 3.2 CATIA Model Conversion to Modelica

Depending on the type of available CAD data different Modelica model structures are produced. CATIA parts (\*.CATPart) are converted to Modelica models and CATIA products (\*.CATProduct) to Modelica packages. Figure 4 shows a CATIA model with two connector frames and the corresponding model structure in Modelica. The Modelica model consists of components from different Modelica libraries parameterized by CAD data. The Modelica MultiBody

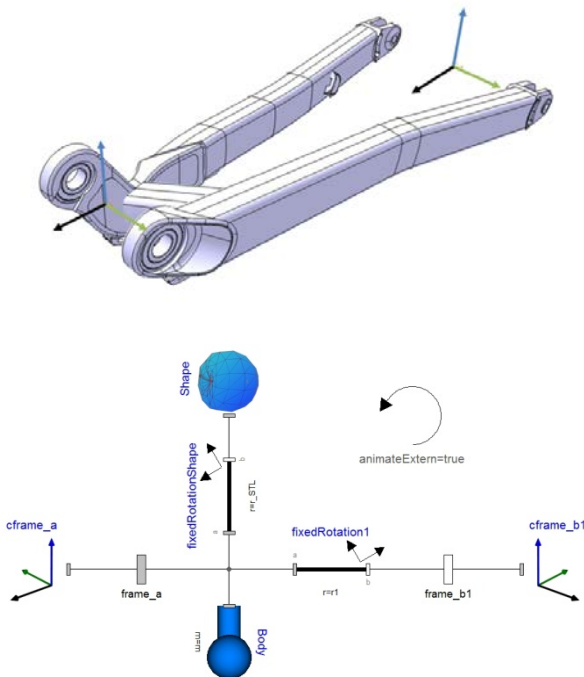


Figure 4: CATIA model (above) and corresponding Modelica model (below)

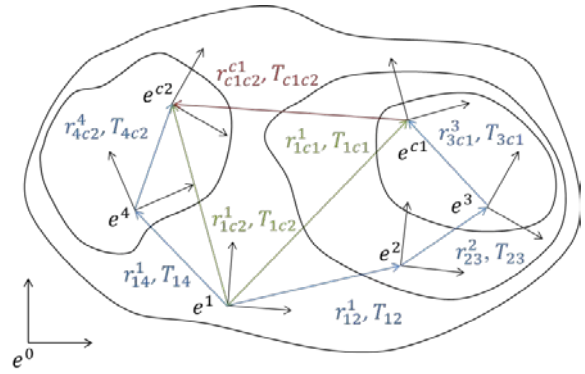


Figure 5: Nested product structure of an assembly in CATIA

Library [5] provides connecting frames (frame\_a, frame\_b1 in Figure 4) that represent the axis systems in the CATIA model. These frames provide cut-forces and cut-torques that are transferred over joints between two components. The first axis system in the connector set of a part or a sub product is set as initial frame\_a for this model. The remaining axis systems are translated to connector frames frame\_b1 to frame\_bn and connected over fixed rotations (fixedRotation) with frame\_a. These models need the relative direction matrix and components of the transformation matrix between the frames as input.

Figure 5 illustrates an example of a nested product structure. Every product has its own body fixed axis system  $e^n$  that can be rotated and moved in relation to the axis system on a higher level. CATIA provides the direction vector  $r_{nn+1}^n$  and the transformation matrix  $T_{nn+1}$  of every component in the related body fixed axis system  $e^n$ . In this case axis system  $e^{c1}$  is chosen as frame\_a. The inputs for the fixed rotation from frame\_a to frame\_b1 are  $r_{c1c2}^{c1}$  and  $T_{c1c2}$ . Equations (1) and (2) allow a recursive transformation of every axis system to  $e^1$ :

$$\begin{aligned} r_{1c1}^1 &= r_{12}^1 + r_{23}^1 + r_{3c1}^1 \\ &= r_{12}^1 + T_{12}^{-1}r_{23}^2 + T_{12}^{-1}T_{23}^{-1}r_{3c1}^3, \end{aligned} \quad (1)$$

$$T_{1c1} = T_{12}T_{23}T_{3c1}. \quad (2)$$

The relative direction vectors and rotations between  $e^{c1}$  and  $e^{ci}$  in axis system  $e^{c1}$  are calculated with the help of equations (3) and (4):

$$r_{c1c2}^1 = r_{c2}^1 - r_{c1}^1, \quad (3)$$

$$r_{c1c2}^{c1} = T_{1c1}r_{c1c2}^1. \quad (4)$$

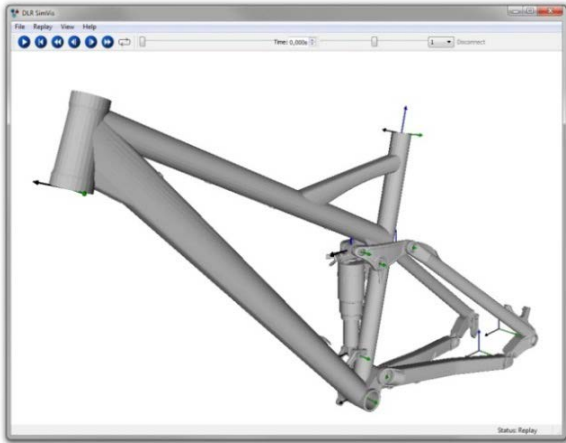


Figure 6: Visualization of a CAD assembly exported from CATIA in DLR SimVis

In addition to the geometric skeleton a body model is introduced containing the mass, a vector to the center of gravity and the body's inertia in respect to the initial frame. If assemblies are translated a model like previously described is generated for every product on top-level that contains connector sets. They are grouped together in a Modelica package.

The visualization is either run internally as Dymola animation with display of the multibody skeleton or externally animated by DLR's SimVis software [6]. SimVis integrates \*.STL (Surface Tessellation Language) files generated from the CAD package as three-dimensional shaded shape representations (Figure 6). Every frame has a visualizer in form of a coordinate system corresponding to the CAD package to simplify simulation work.

Design parameters are integrated as user input parameters on model level. The multibody parameters are either written in the Modelica model as fixed values or stored in a separate textual data file. In the second case the parameters are read into the model over the function `readRealParameter()` to enable parameterized simulation work based on the textual data file.

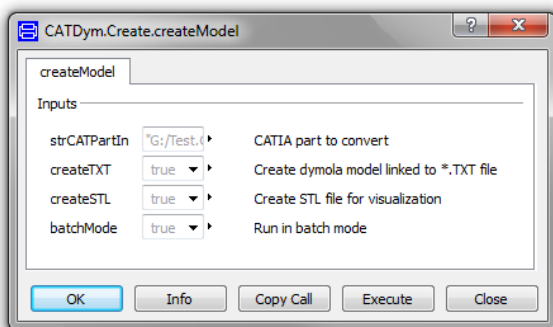


Figure 7: Graphical User Interface for model generation in Dymola

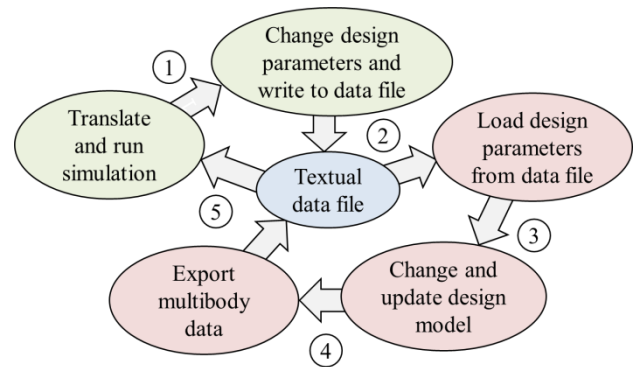


Figure 8: Update process in Dymola (green) and CATIA (red)

The model generation routine is triggered by a Modelica scripting function in Dymola that executes CATIA and runs macros from a VBA Library. Figure 7 shows the Graphical User Interface of the function for model generation with input parameters. The multibody parameters are either stored directly in the model as fixed values or optionally separated from the model in a data file. Furthermore the user can choose to export a three-dimensional shape representation for visualization and activate batch processing. The generated models are manually integrated in simulation experiments by the user in order to investigate the kinematic behavior of the mechanic system. Therefore a multibody structure with joints has to be set up and fitted into testing environments in Dymola.

### 3.3 Updating the Modelica Model Parameters

Throughout the development process changes to the design are applied due to consequences of simulation results or other new requirements. The design model data is either manually modified or changed by design parameters. Isolating the multibody parameters by a text file from the simulation model enables changes to physical and geometrical data without having to reload the model or package in Dymola. Figure 8 leads through the update process and shows the interaction between the different domains. In a first step the design parameter values of the Modelica model are changed in Dymola and written to a textual data file. Then an update routine in the CAD system is triggered by Dymola. The design model is opened in CATIA and the design parameters are read from the data file. After that they are changed in the CATIA design model. The part or assembly is updated in the CAD system and the modified multibody data is exported to the data file analogous to the model generation process in Subsection 3.2.

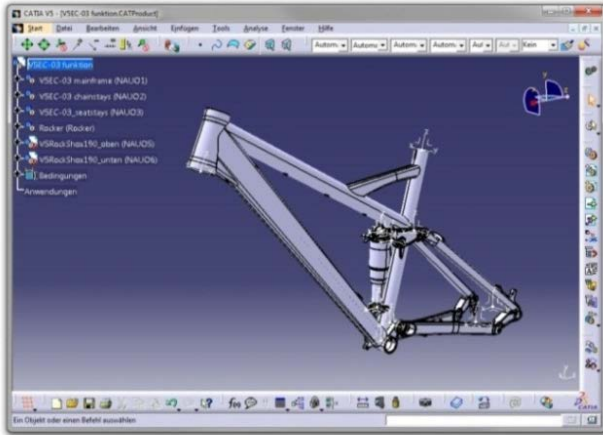


Figure 9: Bicycle rear suspension in CATIA

Finally, the corresponding Modelica simulation model is updated with the multibody parameters through the `readRealParameter()` function reading the parameter values from the text file into the model.

### 3.4 Optimization Loop

The commercial Optimization Library [7] provides several numerical optimization algorithms for solving different kinds of optimization tasks. Multiple tuner parameters with complex optimization criteria allow automated investigations to support the engineering design process. The goal of the optimization task is to minimize user defined criteria with optional equality and inequality constraints. A detailed view on the used optimization and evaluation algorithms can be found in [7]. In order to use the algorithms of the Function Optimization toolbox, the aforementioned update process is implemented in a Modelica function and fitted to the following defined structure:

```
partial function PartialCriteriaVariables
  input Real tuners[:];
  output Real criteria[:];
end PartialCriteriaVariables;
```

In this criteria function, the optimization tuner parameters are mapped to the design parameters defined in CATIA. Based on the tuner values an update routine in CATIA is triggered, followed by a simulation run in Dymola using the changed multibody data as described in chapter 3.3. Then, the simulation results stored in the corresponding `*.mat` file are loaded into workspace through the `readTrajectory()` function. Finally, the values from the loaded

trajectories can be used to formulate the optimization criteria.

## 4 Examples

The following section shows the previously described workflow of model preparation and package generation for an existing bicycle rear suspension from a 2012/13 Fatmodul Ant [8]. After that, the update process is demonstrated in a manual parameter variation followed by an automated optimization utilizing the Optimization Library.

Starting point of the model preparation work is CAD data in `*.STEP` (Standard for the Exchange of Product model data) format. At first the model has to be reorganized with respect to the kinematic structure. The rear suspension is designed as linkage driven single pivot. The shock and the swingarm are connected over additional linkages to generate a progressive leverage ratio between wheel travel and shock absorber travel. The shock, the swingarm (Chain stays) and the upper linkage (Rocker) are mounted in the fixed main frame. The seat stays complete the planar kinematic loop. Figure 9 shows the reorganized model structure on top-level in CATIA. In the next step connector frames are attached to the different parts of the assembly representing the connecting joints between the moving parts. Additionally important load points are defined such as the inertial axis system located in the bottom bracket and the rear dropout. For demonstration purposes a design parameter is defined which refers to the wall thickness of the seat stays. At this point the preparation work in CATIA is finished and the remaining steps are executed in Dymola.

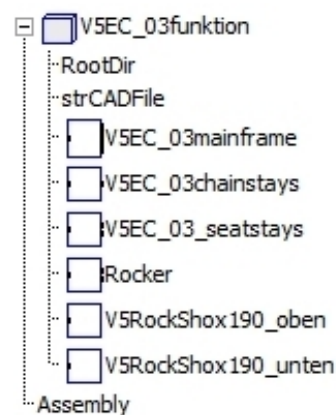


Figure 10: Automatically generated Modelica package of the rear suspension in Dymola

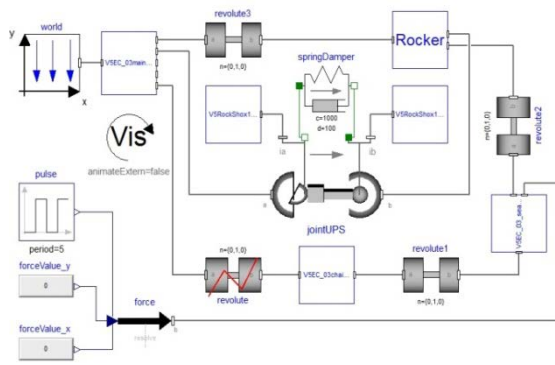


Figure 11: Modelica model of the bicycle rear suspension

A function call in Dymola triggers the package generation in the working directory of the CATIA assembly product. Figure 10 shows the automatically generated Modelica package structure in Dymola. A Modelica counterpart is created for every top-level product from the CATIA design model.

The Modelica package and the associated multibody data file are stored in subfolders of the working directory together with \*.STL visualization data. Based on the kinematic structure a new Modelica model is manually set up by the user to include the generated models. Figure 11 shows the assembled kinematic model of the rear bicycle suspension. The different components of the package are connected via rotational joints with each other. Additional Modelica models complete the overall model. The shock is modeled as springDamper and an input force at the rear dropout is implemented.

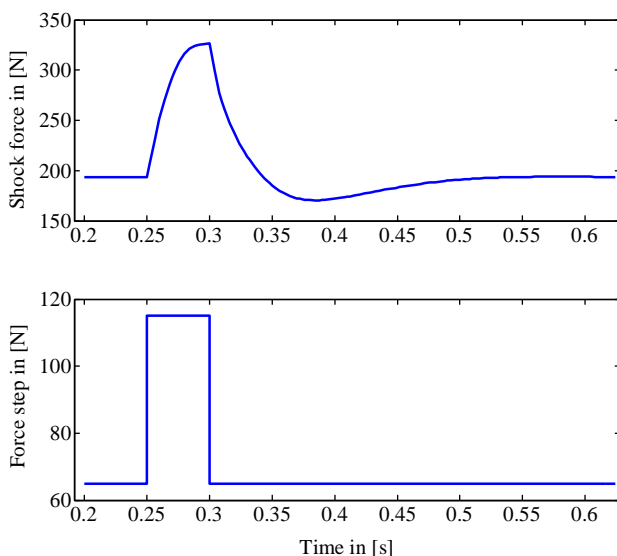


Figure 12: Shock force over time for vertical force step in rear dropout

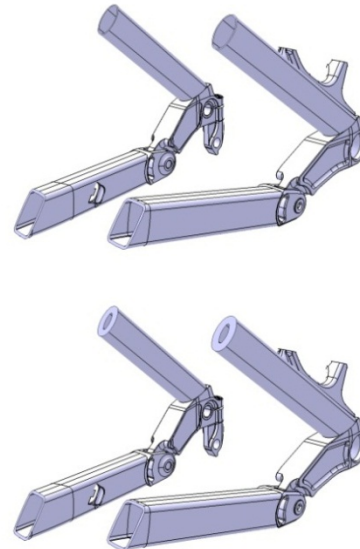


Figure 13: Cut through seat stays with wall thickness  $t = 1 \text{ mm}$  and  $t = 3 \text{ mm}$

Figure 12 shows the step response of the cut force in shock for a force step along the vertical axis of the rear dropout. An overshooting behavior due to predefined spring and damper parameters can be seen. The CAD model consists of 88 parts in 6 top-level products. The Modelica model for this experiment contains 989 components with 963 time-varying variables formulating 18381 equations. The overall process is executed on a X5650 @ 2.67 GHz Workstation with 12 GB RAM. The execution time for the parameter update process in CATIA together with translation and simulation in Dymola based on an already generated Modelica package is 57 s, which contains 29 s for updating the CAD assembly.

Figure 13 shows the inside of the seat stays with different design parameter values. With the help of the update process the wall thickness of the seat stays is changed between 1mm and 6mm in multiple steps. For every loop of the parameter variation the same routine is executed repetitively.

At first, the current design parameter values are written into the corresponding multibody data file. Then the CAD assembly is opened in CATIA and the design parameter values are read from the previously mentioned file. The parameter values are changed in the CAD product structure and the assembly is updated. In the next step, the updated multibody data, such as mass inertias and geometry, is overwritten in the textual data file by CATIA.

Finally, the model is translated in Dymola in order to reread the changed multibody data from the data file and the simulation experiment is run. The simulation

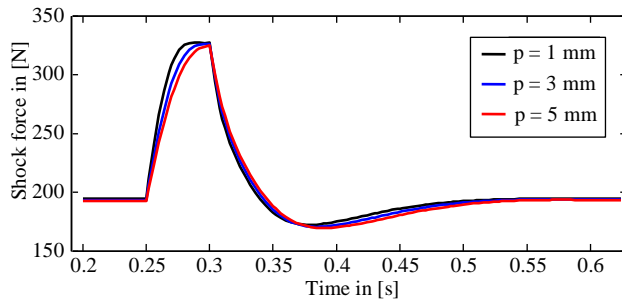


Figure 14: Different shock forces due to variation of wall thickness

results are saved and the next design parameter value is investigated. In this example, no changes to spring stiffness and damping constant were applied during the variation. Figure 14 compares the step responses of the different configurations. Faster response and less overshoot are observed with decreasing wall thickness.

Utilizing the Optimization Library [7], as mentioned in chapter 3.4, allows to investigate the system behavior under complex optimization criteria in comparison to the previously described manual parameter variation. In this example the wall thickness of the seat stays serves as tuner parameter  $p$  and a combined optimization criterion is formulated. The main optimization task is to minimize the overshoot in the shock force, referred to in Figure 15 as  $\Delta f$ . Due to mechanical restrictions in the design of the spring damper system the shock travel  $d$  should not exceed

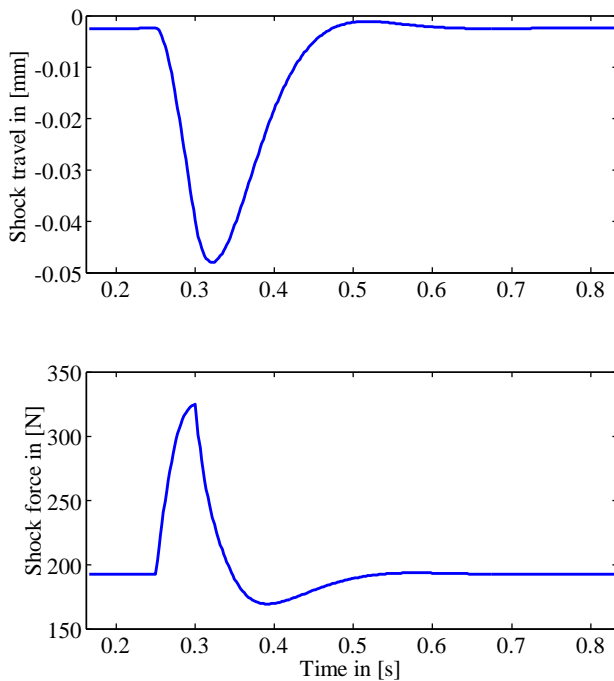


Figure 15: Shock travel  $d$  and overshoot in shock force  $\Delta f$  as optimization criteria

Parameter	Value
Number of criteria evaluations	29
Initial wall thickness $p^0$	3 mm
Range of wall thickness $p$	1 mm – 6 mm
Optimization method	Pattern search
Computation time	1766 s
Best tuner parameter $p^*$	5.43 mm
Overshoot of shock force $\Delta f^*$	23.14726 N
Shock travel $d^*$	47.99986 mm

Table 2: Optimization preferences and results

48 mm. The following formulation summarizes the optimization problem:

$$\min_{1 \leq p \leq 6} \Delta f(p) \quad \text{s.t.} \quad d(p) \leq 48. \quad (5)$$

Table 2 summarizes the results of the optimization run. A solution  $p^*$  is found by a Pattern Search algorithm in less than half of an hour execution time. The solution activates the inequality constraint, i.e.  $d^* \approx 48$  mm and minimizes the overshoot of the shock force.

The shown examples serve mainly for demonstration purpose of the working toolchain and shall illustrate the potential of parameterized design and combined

## 5 Conclusions

An automated model conversion from CATIA to Modelica has been described in this paper. In contrast to existing conversion approaches the paper focuses on the simulation process with Modelica in Dymola. The quick and easy CAD model preparation task can be executed solitary by a simulation expert with basic CATIA experience. The model generation and update toolchain after the model preparation in CATIA is completely controlled out of Dymola. The multibody and design parameters are stored in the model as fixed values or in a separate textual data file. In this way the Modelica models that are derived from CAD are separated from the simulation experiments. Design parameters together with the Optimization Library enable automated parametric design studies. The algorithms and methods in this Library enable the optimization of a part or an assembly in CATIA considering the dynamic behavior of the underlying multibody system.



In the future the focus will be on code enhancement to decrease processing time. Direct communication between Dymola and CATIA via the Active-X-Com interface could improve the model generation process. Further application examples will validate the process quality.

## 6 References

- [1] Modelica Association: "Modelica - A Unified Object-Oriented Language for Systems Modeling", [www.modelica.org](http://www.modelica.org).
- [2] Dassault Systemes AB: CATIA, [www.3ds.com/products-services](http://www.3ds.com/products-services).
- [3] P. Bhattacharya, N. Suyam Welakwe, R. Makanaboyina, A. Chimalakonda, "Integration of CATIA with Modelica," in *The 5th International Modelica Conference, Vienna, Austria, 2006*, pp. 671–675.
- [4] H. Elmqvist, S. E. Mattsson, C. Chapuis, "Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica," in *The 7th International Modelica Conference, Como, Italy, 2009*, pp. 551–560.
- [5] M. Otter, H. Elmqvist, "The New Modelica MultiBody Library," in *The 3rd International Modelica Conference, Linköping, Sweden, 2003*, pp. 311–330.
- [6] T. Bellmann, "Interactive Simulations and advanced Visualization with Modelica," in *The 7th International Modelica Conference, Como, Italy, 2009*, pp. 541–550.
- [7] A. Pfeiffer, "Optimization Library for Interactive Multi-Criteria Optimization Tasks," in *The 9th International Modelica Conference, Munich, Germany, 2012*, pp. 669–679.
- [8] Fatmodul Bicycles: [www.fatmodul.de](http://www.fatmodul.de).



# Modelling elastomer buffers with DyMoRail

Elisabeth Dumont    Werner Maurer  
Institut für Angewandte Mathematik und Physik  
Zürcher Hochschule für Angewandte Wissenschaften  
Technikumstrasse 9, Winterthur, 8401 Switzerland

## Abstract

In this paper a model for elastomer buffers for longitudinal railway vehicle dynamics is presented. This model is part of the more extended DyMoRail library which allows to simulate longitudinal dynamics of entire railway trains. With this library an efficient simulation of complete train compositions in various combinations is possible. The elastomer buffer can be used in combination with other buffer models and couplers in different test scenarios. We present details of our rubber spring model based on the one-dimensional, non-linear rubber spring model proposed by M. Berg [1][2][3]. To illustrate the behavior of the friction force modelled in the latter, it is compared to a diode model for Coulomb friction similar to the one in the Modelica Standard Library. Simulations for 40 J-buffer known as "Miner40" used for freight waggons during shunting at speeds up to 12 km/h are shown. Also shown are simulations of an entire S-Bahn combination with sixteen cars and fifteen elastomer buffers.

*Keywords: mechanics, railway*

## 1 Introduction

The tough competition in the railway industry is forcing operators to continuously set higher quality and comfort standards. Buffers on railway vehicles, as fabricated by Schwab Verkehrstechnik, are no more simple devices but have to be considered Hi-Tech components. They have to be absolutely reliable and present optimal properties to absorb run-up energy from trailing wagons safely. They have to absorb minor impacts, take up slack between locomotive and wagons and bear the load of preceding wagons when pushing. Years ago it was good enough for couplers and buffers to fulfill UIC (International Union of Railways) stan-

dards. But nowadays manufacturers only survive in this competitive market if they are able to offer optimized solutions regarding force, energy absorption, and driving comfort. The manufacturer has to be able to react quickly and flexibly to the challenges of rail operators and rolling stock manufacturers and offer final products and customized solutions with high customer value landlord. Since hardware tests are extremely costly, modeling and simulation step into the optimization process. DyMoRail allows to optimize buffers and couplers in different configurations for safety, energy absorption and costs. Train composition can be chosen by the user.

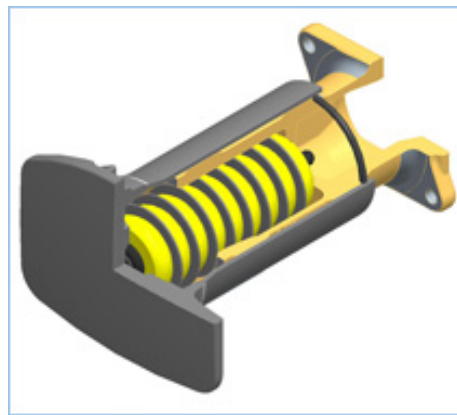


Figure 1: Drawing of a buffer (Seitenpuffer Kategorie A by Schwab Verkehrstechnik).

Schwab Verkehrstechnik AG and ZHAW carried out a project funded by CTI (Swiss Federal Commission for Technology and Innovation) to develop a tool which allows to simulate longitudinal dynamics of entire railway trains. During the following years a Modelica library has been developed, which is called DyMoRail. The DyMoRail library allows an efficient simulation of complete train compositions in various configurations. The library has been presented at Modelica 2012 [4] and a description of the coupler models can be



### 3 Model

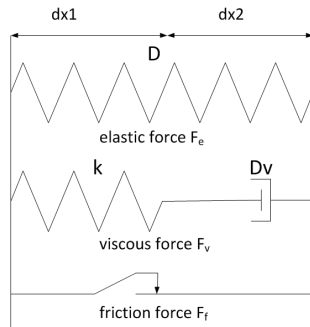


Figure 4: Model for a rubber spring proposed by M. Berg.

For model implementation of the DyMoRail library, the Modelica based simulation software Dy-mola is used. The elastomer model is taken from the non-linear rubber spring model by M. Berg [1][2][3]. The model proposed in these papers is one-dimensional and based on the superposition of three forces (elastic, friction and viscous force) and contains five parameters. For better understanding, a schematic drawing is shown in Figure 4.

In the model by M. Berg, the elastic force is linearly modelled with a stiffness constant  $D$ :

$$F_e = D \cdot x \quad (1)$$

The viscous force is modelled by a linear spring (spring constant  $Dv$ ) in series with a linear viscous damper (damping constant  $k$ ):

$$F_v = Dv \cdot x - kv \quad (2)$$

The friction force  $F_{Coul}$  depends both on the displacement  $x$  and on a reference state ( $x_s, FR_s$ ) in the friction force versus displacement characteristic. Depending on the position relative to this reference state, the friction force is expressed with two parameters maximum force  $FR_{max}$  and constant  $x_2$ . A small value of  $x_2$  gives a steep increase in the friction force and thereby high frictional stiffness. The friction force  $F_{Coul}$  in the model is, depending on how  $x$  is related to the reference displacement  $x_s$ , defined by the equations below.

The reference state is set to  $x_s = 0$  and  $FR_s = 0$ .

$$F_{Coul} = FR_{max} \frac{s_{rel}}{s_{rel} + x_2}$$

For backward movement  $x_s = x_1$  and  $FR_s = FR_1$ :

$$F_{Coul} = FR_1 + (FR_{max} + FR_1) \times \frac{s_{rel} - x_1}{x_2 \left(1 + \frac{FR_1}{FR_{max}}\right) - (s_{rel} - x_1)}$$

For backward movement  $x_s = x_1$  and  $FR_s = FR_1 = FR_{max}$ :

$$F_{Coul} = FR_{max} \left(1 - 2 \frac{s_{rel} - x_1}{2x_2 - (s_{rel} - x_1)}\right)$$

For forward movement  $x_s = x_3$  and  $FR_s = FR_3$ :

$$F_{Coul} = FR_3 + (FR_{max} - FR_3) \times \frac{s_{rel} - x_3}{x_2 \left(1 - \frac{FR_3}{FR_{max}}\right) + (s_{rel} - x_3)}$$

For forward movement  $x_s = x_3$  and  $FR_s = FR_3 = -FR_{max}$ :

$$F_{Coul} = FR_{max} \left(s \frac{s_{rel} - x_3}{2x_2 + (s_{rel} - x_3)} - 1\right)$$

When the buffer is at rest  $x_s = x_5$  and  $FR_s = FR_5$ :

$$F_{Coul} = FR_5 + FR_{max} \frac{s_{rel} - x_5}{x_2 + (s_{rel} - x_5)}$$

When the buffer is at rest mode  $x_s = x_5$  and  $FR_s = FR_5 = \pm FR_{max}$ :

$$F_{Coul} = FR_{max} \left(\pm 1 + \frac{s_{rel} - x_5}{x_2 + (s_{rel} - x_5)}\right)$$

The implementation of these equations in Modelica is as follows

```

dx = dx1 + dx2
dv2 = der(dx2)
k * dv2 = Dv * dx1
Vorzeichen = if dv > 0
then 1
else if dv < 0
then -1
else 0
FCoul = FCouls +
(dx - dxs) * (FCoulmax - Vorzeichen * FCouls)
/(x2 * (1 - Vorzeichen *
FCouls/FCoulmax)
+ Vorzeichen * (dx - dxs))
when change(Vorzeichen) then
dxs = pre(dx)
FCouls = pre(FCoul)
end when
F = D * dx + Dv * dx1 + FCoul
der(W) = P
    
```

where  $dx_1$  is the displacement of spring  $Dv$  and  $dx_2$  is the displacement of the damper  $k$ .

To illustrate the behavior of the M. Berg friction model, it (Figure 5) is compared to a diode model for Coulomb friction, similar to the one of the Modelica Standard Library (Figure 6), as described for example in reference [7]. As can be seen from these graphs, the hysteresis loop of the Modelica Standard Library model has a rectangular form, whereas the M. Berg friction model shows sharp corners at the maximum and minimum displacements with a smooth transition between the upper and lower branch of the hysteresis.

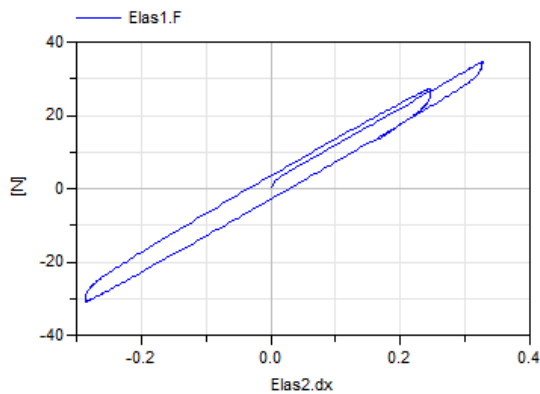


Figure 5: Friction force versus displacement. Simulation of the friction model for a rubber spring proposed by M. Berg.  $F_{max} = 2$  and  $x_2 = 0.002$ .

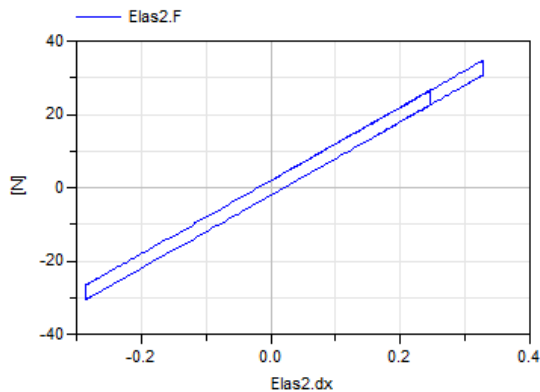


Figure 6: Friction force versus displacement. Simulation of the friction model for the basic model for Coulomb friction in the Modelica Standard Library.  $F_{max} = 2$  and  $x_2 = 0.002$ .

In the final model for the Miner buffer in the DyMoRail library, the total force:

$$F = F_e + F_v + F_{Coul} \quad (3)$$

is taken to the exponent of an exponential function. This corresponds more to reality, shown by a thorough parameter study.

## 4 Simulation

For their acceptance test, railway companies define several crash scenarios with different crash partners at different speeds.

Figure 7 shows simulations for 40 J-buffer known as "Miner40" at different initial speeds. This type of buffers is exclusively used for freight wagons. The acceptance tests demand reversible shunting at speeds up to 12 km/h. During collisions at low speeds, the energy has to be absorbed by the buffers. Absolutely no damage should occur to the rolling stock. The acceleration must remain below 4.0 g.

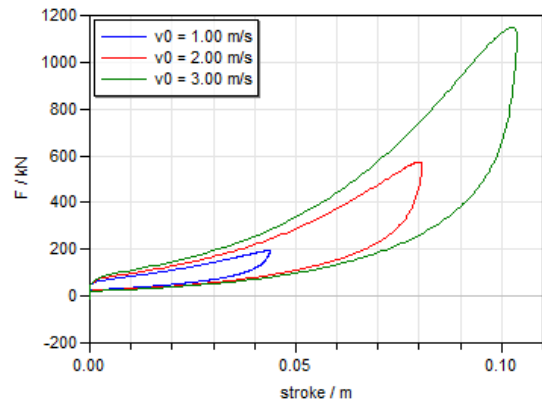


Figure 7: Simulation of the buffer model with different initial speeds. Force vs stroke diagram.

The elastomer model works also in combination with other coupler or buffer models in complete train configuration. Figure 8 shows a simulation of three S-Bahn combinations colliding with a single combination at rest during shunting. Each S-Bahn contains four cars and three Miner models, so in total the model contains sixteen cars and fifteen buffer models. In this graph, one can distinguish the different cars colliding one after the other.

## 5 Conclusion

In this paper a model for elastomer buffers for railway vehicle dynamics is presented. This model is part of the more extended DyMoRail library which allows to simulate longitudinal dynamics of entire

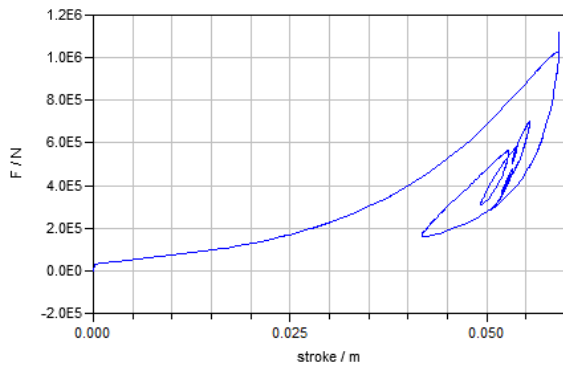


Figure 8: Simulation of three S-Bahn combinations colliding with a single combination at rest during shunting.

railway trains. With this library an efficient simulation of complete train compositions in various combinations is possible. The elastomer buffer can be used in combination with other buffer and couplers models. The acceptance test scenarios can be simulated with the help of this model.

## 6 Acknowledgements

This project has been funded by CTI (Swiss Federal Commission for Technology and Innovation, contract number 14165.1).

## References

- [1] Berg, M. A model for rubber springs in the dynamic analysis of rail vehicles, Proc Instn Mech Engrs, Vol 211 Part F (1997), pp. 95-108.
- [2] Berg, M. A Non-Linear Rubber Spring Model for Vehicle Dynamics Analysis, Vehicle System Dynamics Supplement, Swets and Zeitlinger Publishers, 28 (1998), pp. 723-728.
- [3] Berg, M. A Non-Linear Rubber Spring Model for Rail Vehicle Dynamics Analysis, Vehicle System Dynamics, Swets and Zeitlinger Publishers, 30 (1998), pp 197 - 212.
- [4] Dumont E. and Maurer W. Proceedings of the 9th International Modelica Conference, 2012, p.691-696.
- [5] Maurer W. Puffer nach Mass. Eisenbahn Revue 3, 2003, p.118-119.

- [6] Maurer W. Simulationsgestützte Entwicklung von Puffern und Dämpfern für Eisenbahnzüge. Proceedings of the 18th Symposium on Simulationstechnik ASIM 2005, Erlangen, Germany, ASIM September 12-15 2005.
- [7] Otter M., Elmqvist H., Mattsson S.-E. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle, Proceedings of the 1999 IEEE Symposium on Computer Aided Control System Design (CACSD'99), Hawaii, Aug. 22-27, 1999.





# A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces

Felix Oestersötebier

Peng Wang

Ansgar Trächtler

Control Engineering and Mechatronics, Heinz Nixdorf Institute, University of Paderborn

Fürstenallee 11, 33102 Paderborn, Germany

{felix.oestersoetebier, peng.wang, ansgar.traechtler}@hni.uni-paderborn.de

## Abstract

Modeling of multibody mechanics plays a central role in the design of mechatronic systems. In technical use-cases, they often contain loose couplings, where contact is possible. We present a ready-to-use contact library in Modelica. It comprises surface definitions for simple contact surfaces, which can be connected with multibodies of the Modelica Standard Library. It implements a force-based approach between single contact points. The contact forces are calculated in configurable non-central contact blocks. Exemplarily, the results of three experiments are shown and compared to benchmark simulations.

*Keywords:* contact library; simple contact surfaces; non-central contact block; contact forces

## 1 Introduction and Motivation

In the design process of mechatronic systems, the designers are facing the challenge of developing and controlling the more and more complex dynamics of the system. Therefore, multi-domain simulation models come to use from the outset of the conceptual design stage. As this also denotes a significant modeling effort, object-oriented modeling languages like Modelica offer the possibility to utilize and/or build up model libraries. The concept of ports makes it possible to combine and simulate components of different domains and origins in one model of the system. We intend to extend the available libraries by providing an idealized contact library that makes it possible to model contact phenomena. In particular, it should be possible to define each component separately in order to be able to reuse and combine approved patterns in new applications (c.f. [1]).

Considering technical use-cases one often finds multibody mechanics that typically comprise mainly fixed (e.g. kinematic chains) and few loose

couplings, where dynamic contact phenomena take place. Nevertheless, modeling contacts is “a key factor and a challenging problem in simulation of multibody systems (MBS), where a balance between performance and accuracy has to be found” [2]. However, to the best of our knowledge, there is currently no ready-to-use Modelica library available to handle contact problems in any level of detail. Otter et al. [3] suggest a force-based extension to the Modelica MultiBody Library to enable central collision handling. Herein, three variants to define contact surfaces are described (parametric surfaces, algebraic constraint surfaces, surfaces of polytopes). In [4] the surfaces of arbitrary bodies in the MBS are discretized by means of polygons. The approach presented in this paper differs from that, as it implements contact modeling by means of non-central contact blocks. It provides combinable, simple contact surfaces, which are described by single contact points.

We observed that in many industrial applications, even if the complete bodies are of complex shape, only a certain part of them contacts with others. The actual contact surfaces are often designed to be simple. Thus, the aim is to enable the designer to perform simulations of such systems including idealized representation of the contacts. Thereby, the analysis of the principle functional capability of the system in the course of the conceptual design is focused. This entails specialized modeling principles concerning the usability and the interpretation of the simulation results, which both should be relatively easy.

## 2 Concept of the Contact Library

Multibody mechanics is usually modeled using rigid bodies, which are described by their mass, located in the center of mass (CM), and their moment of inertia. In order to be able to model elastic, non-stiff collisions one has to weaken the rigid body assumption a little. Furthermore, one has to consider

the shape of the bodies. We address these two issues in the following subsections.

### 2.1 Idealization of the Elastic Foundation Model

References in literature introduce a so called “foundation model” to approximate the complex contact theory of Hertz and others in the context of multibody simulation [4,5,6,7,8]. It is assumed that the contact area is small compared to the dimensions of the contacting bodies (nonconforming contacts). The foundation model comprises a thin elastic or viscoelastic layer between the rigid bodies and neglects shear stress. With these assumptions, which are valid for isotropic and homogenous materials, one is able to describe dynamic as well as static contact incidents. The force-based approach contrasts with idealized impulse-based calculations, which are only applicable for stiff contacts [3].

We use a further idealization (c.f. Figure 1) that is based on single force elements. Assuming that the contact area is not only relatively small but also of an idealized shape, we describe it by means of single contact points. A nonlinear spring-damper element is inserted to calculate the normal force  $F_n$  between these points of the colliding surfaces. This requires the previous identification of possible contact points on the rigid body surfaces and the continuous determination of the normal direction. For these purposes, we provide analytic solutions for simple geometries in our library. The normal force is then used to determine the friction force  $F_t$  between the two bodies in the tangential direction.

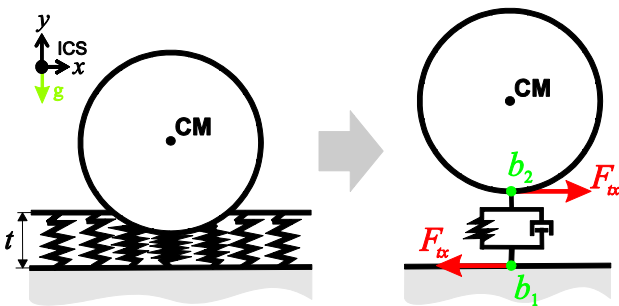


Figure 1: Idealization of the elastic layer

### 2.2 Classification of Contacts

As a starting point, we focused on *spherical*, *cylindrical* and *plane* surfaces, either in *rectangular* or *circular* shape. Depending on the shape of the contact area, we use 1 (punctiform), 2 (linear) or 4-5 (planar) points to describe it. The position of the respective number of *potential contact points* is calculated on both of the contact bodies and the collision detection is performed for each pair of

potential contact points. Figure 2 shows the shape of the contact area for all possible combinations, as well as the number of contact points. As the contact region may alter with the moving bodies, the contact points will also move on the defined surface.

	<b>punctiform</b> (1 point)	<b>punctiform</b> (1 point)	<b>punctiform</b> (1 point)
	<b>punctiform</b> (1 point)	<b>punctiform/linear</b> (1/2 points)	<b>linear</b> (2 points)
	<b>punctiform</b> (1 point)	<b>linear</b> (2 points)	<b>planar</b> (4-5 points)

Figure 2: Shapes of the contact area and number of contact points for the different contact pairs<sup>1</sup>.

### 2.3 Definition of Contact Surfaces

To describe the aforementioned, elementary contact surfaces (Figure 2), we provide ready-to-use blocks. The surface blocks represent a thin and massless layer, which can be connected to any kind of rigid body by a *frame* connector (c.f. Figure 3). The dimensions of the surface can be parameterized. In order to be able to use the dimensions in the contact block we introduce a new interface to connect the surface definition with a contact block. This adds a vector to the MultiBody *frame* of the Modelica Standard Library that comprises maximum three terms to describe the surface geometry (e.g. width, length) and the respective direction vectors to get the orientation in the connected *frame*. The latter constitutes the body coordinate system (BCS) of the contact surface.

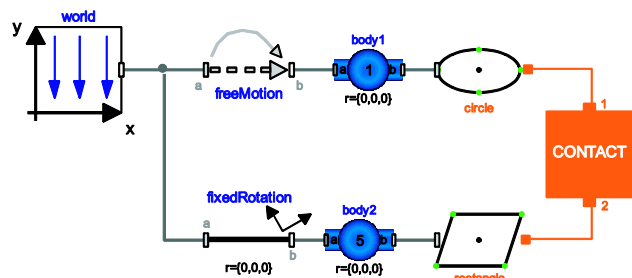


Figure 3: Example of a multibody model including contact

<sup>1</sup> Spherical, cylindrical and plane surfaces are displayed by the respective icon, which also indicates the maximum number of contact points.

Figure 3 displays an example of the intended use of the contact library. While the shape of the rigid bodies may be arbitrary, the shapes of the surfaces that possibly can collide are defined to be a circular and a rectangular plane. For example, body 1 may represent a bottle with a round and plane base. As this stands on a rectangular desk only these two simple contact surfaces have to be defined. More complex contact surfaces may be assembled from elementary ones, where every pair of surfaces is connected via a contact block. The contact block is adjusted to the respective surface combination by using the *replaceable* method for class parameterization in Modelica.

### 3 Modeling of the Contact Block

The calculation of the appropriate force in the contact block clearly depends on the combination of surfaces. Nevertheless, a comprehensive sequence (c.f. Figure 4) can be established that is implemented in the contact block and performed in each time step. As mentioned before, the *sphere-to-sphere*, *sphere-to-cylinder* and *sphere-to-plane* contact areas are described via a punctiform contact. In these cases, three steps have to be passed to determine the contact force. First, a potential contact point is determined on each of the two contact surfaces. The contact condition for these two points is checked in the next step. If the condition is fulfilled, the two bodies collide and the contact force between the two contact points is calculated. Otherwise, no contact force is applied.

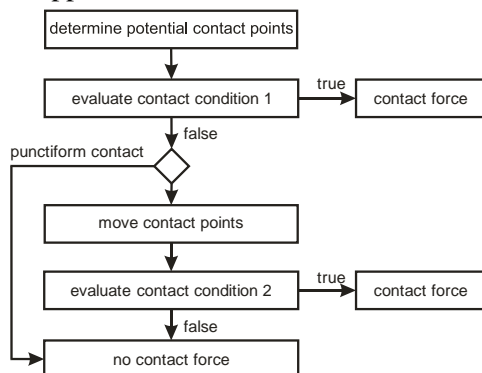


Figure 4: Calculation of the contact force

In the case of linear or plane contact, the contact area may become smaller if only parts of the surfaces collide, whereas the shape stays the same. Thus, additional contact point movement may be necessary. Here, the detection of contact points and the contact condition checking are performed analogously in the first two steps. If the contact condition 1 is not fulfilled, the contact points are

displaced to a new position on the contact surfaces in a way that contact between the new potential contact points might be possible. The latter is checked by evaluating the second contact condition. If a collision occurs, the respective contact force is calculated and applied.

#### 3.1 Contact Detection

The main principle of the implemented contact detection is shown with the help of two examples. The first example comprises a contact between a sphere and a cylinder (c.f. Figure 5). Initially, the bodies are represented by their centroids  $a_1$  and  $a_2$  in the MBS model. Two body-fixed frames BCS1 and BCS2 describe these two points. Furthermore, we assume that the lateral surface of the cylinder and the spherical surface of the ball are defined according to Section 2.3. Consequently, the two frames, the radius  $r$  of the ball and the length  $L$  and the diameter  $D$  of the cylinder, as well as the direction vectors of  $L$  and  $D$  resolved in BCS1, are available in the contact block.

In order to determine the potential contact points, the sphere-centroid  $a_2$  is projected on the longitudinal direction of the cylinder. The latter is represented by the  $x$ -direction of BCS1 by default. The result of the projection is the point  $a'_2$  given in BCS1. However, the absolute position of the point  $a'_2$  is limited to  $\pm L/2$ . This ensures that the point is always located between the two end faces of the cylinder. The local coordinate system LCS1 is defined in  $a'_2$  and has the same orientation as BCS1.

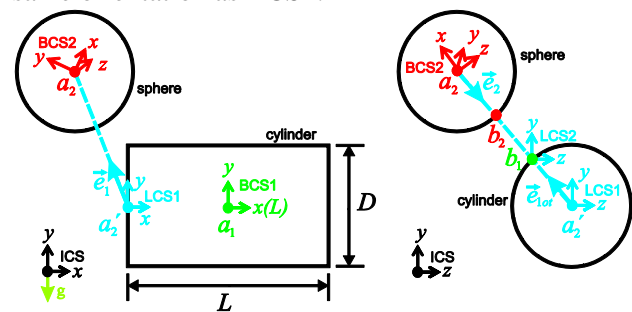


Figure 5: Contact point detection of the *sphere-to-cylinder* contact model

Based on vector  $\overline{a'_2 a_2}$  the direction vector  $\vec{e}_1$  is computed. Then, the orthogonal projection  $\vec{e}_{1ot}$  of the vector  $\vec{e}_1$  is determined in the  $y$ - $z$ -plane of LCS1. We obtain the first potential contact point  $b_1$  by displacing  $a'_2$  along the vector  $\vec{e}_{1ot}$ . As  $b_1$  lies on the cylinder surface, the distance is given by  $D/2$ . In the potential contact point  $b_1$ , the local coordinate system LCS2 is defined. Again, the orientation of it is equivalent to BCS1. With the help of LCS2 the position of the potential contact point  $b_2$  on the ball-

surface can be calculated. Therefore, the vector  $\overline{a_2 b_1}$  and the direction vector  $\vec{e}_2$  are determined. By moving  $a_2$  along  $r\vec{e}_2$  we calculate the position of the second potential contact point  $b_2$  on the spherical surface. The vector  $\vec{e}_2$  also constitutes the normal direction for the force calculation (c.f. Section 3.3) in this case.

After the determination of the potential contact points  $b_1$  and  $b_2$ , the contact condition is evaluated (c.f. Equation (1)). It consists of two terms. On the one hand, it is verified that the distance between the points  $b_2$  and  $a_2'$  is less than radius of the cylinder end face ( $D/2$ ) and greater than the difference between the radius and the maximum penetration  $t$ . The latter results from the assumption of a thin contact layer. This evaluation is performed in LCS1. On the other hand, the interval  $|x_{b_2}|$  between  $b_2$  and  $a_1$  in the longitudinal direction of the cylinder has to be less than or equal to  $L/2$ . This is evaluated in BCS1. If both conditions are fulfilled at the same time, the two bodies intrude and the contact force is applied to the contact points (c.f. Section 3.3).

$$\text{contact} = \left( \frac{D}{2} - t \leq |\overline{a_2' b_2}| \leq \frac{D}{2} \right) \cap \left( |\vec{e}_{x_{BCS1}} \cdot \overline{a_1 b_2}| = |x_{b_2}| \leq \frac{L}{2} \right) \quad (1)$$

The second example is the collision between a cylinder and a circular plane (see Figure 6), which denotes a linear shape of the contact area. As regards the circular plane, the geometry is sufficiently described by the radius  $R$ . Again, body-fixed coordinate systems (BCS1 and BCS2) are defined in the centroids  $a_1$  of the cylinder and  $a_2$  of the plane. The  $y$ -direction of BCS2 represents the vertical direction of the plane and the normal direction for force calculation. According to Figure 2 the contact area has a linear shape. Thus, two potential contact points have to be determined on the surface of the cylinder. These do not necessarily touch the plane at the same time.

To detect the contact points on the surface of the cylinder, the centers  $a_{1l}$  and  $a_{1r}$  of the two end faces are observed. The local coordinate systems LCS1 and LCS2 in these points are oriented like BCS2. In preparation for the potential contact point  $b_{11}$ , the unit vector  $\vec{e}_L$  of the  $x$ -direction in BCS1 and the orthogonal projection  $\vec{e}_{Lp}$  on the  $x$ - $z$ -plane of BCS2 are determined. Then, we implemented the following rotation sequence to get local coordinate systems that are orientated as needed. As regards  $a_{1l}$ , the local coordinate system LCS1 is rotated about the  $y$ -axis (LCS'1) and the  $x$ -axis (LCS''1) thereafter. The rotation angle  $\alpha$  is determined by the scalar product of  $\vec{e}_{Lp}$  and the unit vector of the  $x$ -direction of the

BCS2. Whereas  $\beta$  is specified by the angle between the vectors  $\vec{e}_L$  and  $\vec{e}_{Lp}$ . The contact point  $b_{11}$  is then given by  $(0, -D/2, 0)$  in the obtained system LCS''1. With the same principle, one can also determine the point  $b_{12}$ . In addition, the contact points  $b_{21}$  and  $b_{22}$  on the surface of the circular plane can be located by projecting  $b_{11}$  and  $b_{12}$  onto the plate (BCS2).

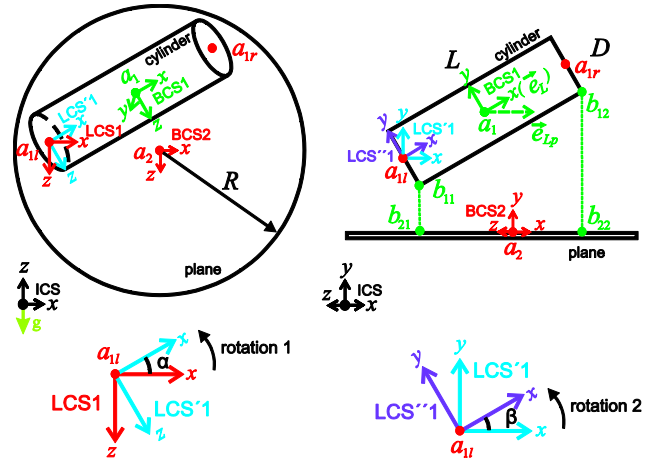


Figure 6: Contact point detection of the circle-to-cylinder contact model

Based on these preliminary, potential contact points the contact condition 1 is evaluated. It consists of three parts. As an example, the condition related to  $b_{11}$  is given in Equation (2). The variables  $x_{b_{11}}$ ,  $y_{b_{11}}$  and  $z_{b_{11}}$  are the components of  $b_{11}$  in the corresponding direction given in BCS2. To detect a collision, the absolute position in the  $x$ - and  $z$ -direction must not be greater than the radius of the plane, and the penetration depth must not exceed the maximum value of  $t$  in the negative  $y$ -direction.

$$\text{contact1} = \underbrace{(|x_{b_{11}}| \leq R)}_i \cap \underbrace{(-t \leq y_{b_{11}} \leq 0)}_{ii} \cap \underbrace{(|z_{b_{11}}| \leq R)}_{iii} \quad (2)$$

### 3.2 Contact Point Movement

However, contact between the cylinder and the circular plane might be possible, even if the contact condition 1 is not fulfilled. Figure 7 shows a configuration where the term  $i$  in Equation (2) is false. In this case, the preliminary contact point  $b_{11}$  should be displaced along the contact surface. We use the law of cosines to calculate the position of this new point  $b'_{11}$ . Considering the displayed triangle (Figure 7), we get the displacement  $b$  by the following equations.

$$R^2 = a^2 + b^2 - 2ab \cdot \cos \gamma \quad (3)$$

$$a = |\overline{b_{11} a_2}|$$

$$\gamma = \angle(\overline{b_{11} a_2}, \overline{b_{11} b_{12}})$$

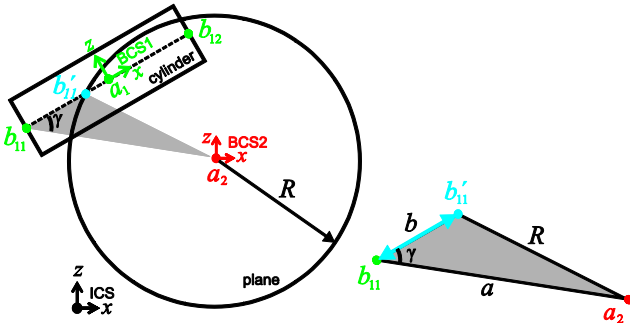


Figure 7: Contact point movement in the *circle-to-cylinder* contact model

The vector  $\overline{b_{11}b_{12}}$  can be replaced by the length direction of the cylinder, which is available from the contact interface (c.f. Section 2.3). Thus, the new contact point  $b'_{11}$  is determined and the analogous contact condition 2 is checked.

$$\text{contact2} = (|x_{b'_{11}}| \leq R) \cap (-t \leq y_{b'_{11}} \leq 0) \cap (|z_{b'_{11}}| \leq R) \quad (4)$$

Since every linear contact region denotes a possible one-directional displacement of preliminary contact points, the contact detection in these cases is very similar. In contrast, planar contact regions lead to two directions of motion. Exemplarily, the contact point movement in the case of two rectangular planes is described in the following.

The initial situation of the example is shown in Figure 8. Again, the body-fixed coordinate systems BCS1 and BCS2 are defined in the two centers  $a_1$  and  $a_2$ . The geometry information of each plane contains the length  $L$  and the width  $W$ . The potential contact points are placed in the four corners of the planes. The contact condition in Equation (5) is used for the potential contact point  $b_{11}$ .

$$\text{contact1} = \underbrace{(|x_{b_{11}}| \leq \frac{L_2}{2})}_i \cap \underbrace{(-t \leq y_{b_{11}} \leq 0)}_{ii} \cap \underbrace{(|z_{b_{11}}| \leq \frac{W_2}{2})}_{iii} \quad (5)$$

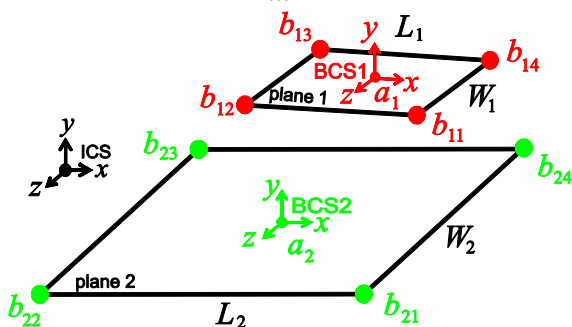


Figure 8: Initial contact points and coordinate systems of the *rectangle-to-rectangle* contact model

If the contact condition in Equation (5) cannot be fully met, the corresponding point is displaced along

the two adjacent edges of the rectangular plane and two new points are obtained. However, it must be ensured that the displacement cannot exceed the length of the respective edge. Figure 9 exemplarily shows a configuration, where one of the initial contact points of plane 1 is outside plane 2. The condition  $i$  in Equation (5) is violated, which means that the contact point exceeds plane 2 in  $x$ -direction of BCS2. The height  $h_{ol}$  of the displayed triangle constitutes the distance between  $b_{11}$  and the nearest edge on plane 2. It is calculated by the following equation.

$$h_{ol} = |x_{b_{11}}| - \frac{L_2}{2} \quad (6)$$

The relationship between the displacement of  $b_{11}$  and the distance  $h_{ol}$  can then be expressed with the help of the angle  $\varphi$ .

$$V_{lw} = \min\left(\frac{h_{ol}}{|\sin \varphi|}, W_1\right)$$

$$V_{ll} = \min\left(\frac{h_{ol}}{|\cos \varphi|}, L_1\right) \quad (7)$$

From these equations the two new potential contact points  $b_{11l}$  and  $b_{11w}$  are obtained by circumventing singularity problems in case of  $\varphi = 0$  or  $\varphi = \pi/2$ . Again, the second collision detection is evaluated to check whether contact forces have to be applied or not. As a result, it is possible that a maximum of five points can represent planar contact areas.

When implementing the contact point movement in Modelica, we tried to minimize the number of DAEs and events. Therefore, for example the *prismatic* block of the Modelica Standard Library has been modified in order to be adaptable to the respective direction. It is furthermore possible to disable the inserted filtering of the contact point displacement. Because it is usually continuously differentiable, this denotes an effective possibility to reduce the simulation time very often (see Section 4).

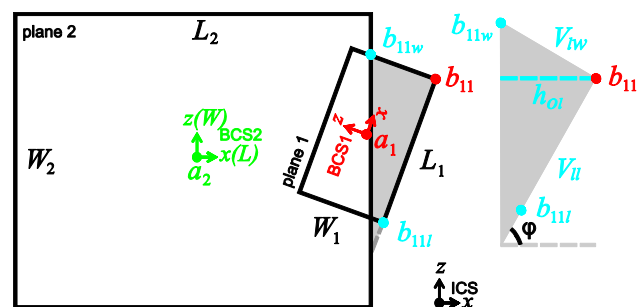


Figure 9: Contact point movement of the *rectangle-to-rectangle* contact model

### 3.3 Contact Forces

The three-dimensional contact force is applied, if the contact condition holds for one contact point. It

consists of both the normal force and the tangential friction force. The respective directions can be obtained by means of the local coordinate systems in the contact points. As aforementioned, the continuous surface layer is replaced by a nonlinear spring-damper element. Consequently, the normal force  $F_n$  is determined by means of the penetration  $p$  in normal direction, and the penetration velocity  $\dot{p}$ . A continuous contact force model with hysteresis damping according to [9] is implemented (Equation (8)). Nevertheless, selecting  $n = 1$ ,  $m = 0$  one can get the linear Kelvin-Voigt model, where the coefficients  $c$  and  $d$  are the spring and damping constants. Choosing  $n = m$  a formulation according to [10] is obtained.

$$F_n = cp^n + dp^m\dot{p} \quad (8)$$

In order to calculate the friction forces without further discontinuous events, which would decrease the simulation speed and impede controller design, we use the continuously differentiable friction model of Makkar et al. [11]. They introduced the following function of the relative velocity  $v_{rel}$  to approximate the friction coefficient  $\mu$  of the characteristic Stribeck curve.

$$\mu(v_{rel}) = \gamma_1(\tanh(\gamma_2 v_{rel}) - \tanh(\gamma_3 v_{rel})) + \gamma_4 \tanh(\gamma_5 v_{rel}) + \gamma_6 v_{rel} \quad (9)$$

Thus, no ideal static friction can be obtained, because the actual force to be applied in the ideal static state is independent of the relative velocity  $v_{rel}$  of the two contact points. Static friction is rather represented by sliding with very small relative velocities. However, this in fact matches the actual characteristics of many tribological systems [12]. To set the unknown, non-physical constants  $\gamma_i$  ( $i = 1 \dots 6$ ) we use five parameters, which are shown in Figure 10. The parameters  $\mu_s$  and  $\mu_k$  denote the coefficients of static and kinetic friction. The limit velocities  $v_{\varepsilon 1}$  and  $v_{\varepsilon 2}$  define the beginning of mixed and viscous friction. The latter is described by the proportionality factor  $k_v$ .

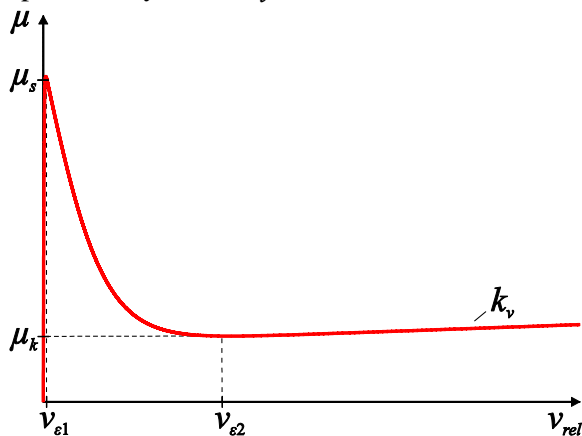


Figure 10: Approximation of the Stribeck friction curve

As an example, the following force vector (resolved in the LCS) is applied to the contact points of plane 1 of the *rectangle-to-rectangle* contact, if the contact condition is fulfilled.

$$F_{contact} = \begin{pmatrix} \mu_x \cdot F_n \\ F_n \\ \mu_z \cdot F_n \end{pmatrix} \quad (10)$$

## 4 Simulation Results

In this section, we will show some of the obtained results. We present three experiments, which were performed in Dymola using the DASSL solver. The results are compared to a benchmark simulation in the commercial MBS software RecurDyn. Herein, a powerful recursive algorithm to model contact problems is implemented, which is based on contact forces as well [13]. Despite the comprehensive and complex *solid-to-solid* contact that can be used for arbitrary CAD-geometries, RecurDyn offers the possibility to utilize idealized contact definitions for simple surfaces. In each of the three experiments, all parameters<sup>2</sup>, including the contact force calculation as well as the solver settings, are attuned to fit each other exactly. The direction of gravity is the negative  $y$ -direction of the ICS. When selecting the presented experiments, we refer to the aforementioned surface configurations (c.f. Section 3).

*Experiment 1* comprises a sphere ( $r = 0.015$ ,  $m = 0.11$ ) falling onto a cylindrical contact surface. The initial position of the sphere centroid is  $(0, 0.05, 0.002)$ . It is not coupled, whereas the position of the cylinder is fixed in the ICS. The latter has a diameter of  $D = 0.015$  and a length of  $L = 0.05$ . In this configuration two collisions occur. After the second contact with the cylinder surface, the sphere falls down beside the cylinder. Figure 11 shows the  $y$ -position of the centroid of the sphere. One can see that the calculated trajectories of the different tools are comparable. In this case, the effect of the nonlinear damping seems to be slightly higher in Dymola than in RecurDyn. Nevertheless, the peaks in the normal forces nearly coincide at approximately 21 N and 9 N (not shown).

In *Experiment 2*, a cylindrical body ( $d = 0.015$ ,  $l = 0.03$ ,  $m = 0.04$ ) falls on a circular plane ( $d = 0.08$ ) and then rolls down. The plane is fixed in the ICS, but rotated by  $5^\circ$  around the  $x$ -axis. The initial position of the cylinder is  $(0, 0.02, 0)$ . As the cylinder reaches the end of the plane, the preliminary contact points are displaced along the length

<sup>2</sup> All units are specified according to the International System of Units (SI)

direction. In Figure 12 some results are displayed exemplarily. We again observe that the positions of the centroid largely comply. The initial angular velocity (around the length direction) of the cylinder after the hit is slightly smaller in RecurDyn than in Dymola, while the acceleration is equivalent. The RecurDyn model (solid-to-solid contact) shows some implausible behavior right before the end of the plane is reached. Unless no external force is applied, deceleration can be monitored here. For these two reasons, the cylinder stays about 10ms longer on top of the plane.

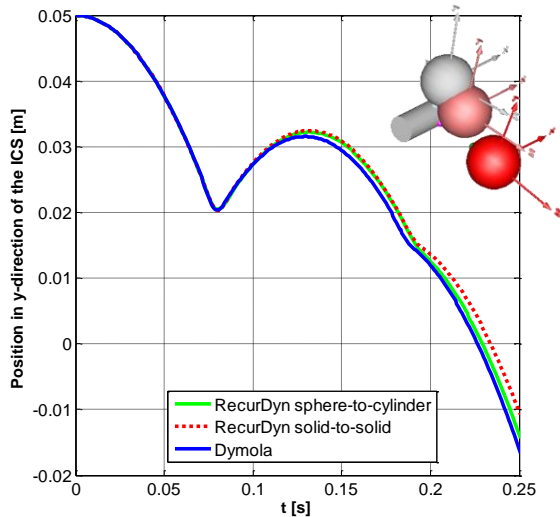


Figure 11: Sphere falling on a fixed cylinder; parameters:  $c = 1e^5$ ,  $d = 10$ ,  $n = 1.5$ ,  $m = 1.5$ ,  $t = 5e^{-3}$ ,  $\mu_s = 0.03$ ,  $\mu_k = 0.02$ ,  $v_{\varepsilon 1} = 0.01$ ,  $v_{\varepsilon 2} = 0.1$ ,  $k_v = 0$

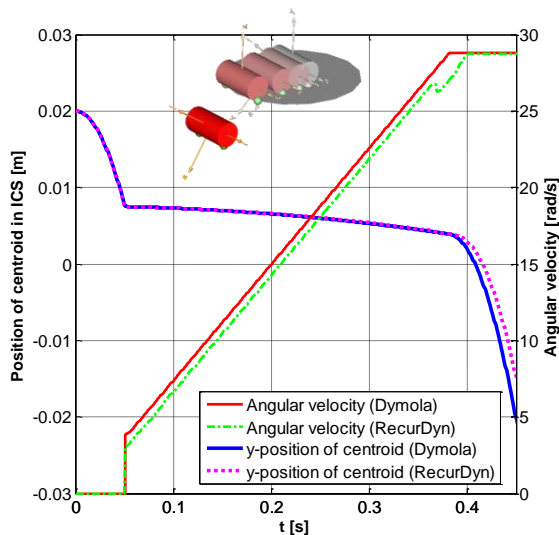


Figure 12: Cylinder on a circular plane; parameters:  $c = 5e^3$ ,  $d = 5e^2$ ,  $n = 1$ ,  $m = 0$ ,  $t = 3e^{-3}$ ,  $\mu_{s,k} = 0.12$ ,  $v_{\varepsilon 1,2} = 0.01$ ,  $k_v = 0$

*Experiment 3* contains a cuboid body ( $l = 0.07$ ,  $w = 0.1$ ,  $h = 0.05$ ,  $m = 2.75$ ) falling and afterwards sliding on a rectangular plane. The plane

( $l = 0.2$ ,  $w = 0.25$ ) is again fixed in the ICS and rotated by  $15^\circ$  around the z-axis. What can be seen in the plots (Figure 13) is that we get very similar movement of the box, despite an observed higher velocity in RecurDyn. In our opinion, the small divergence we obtain may be due to three reasons. (1) Again, the effect of the damping is slightly higher in Dymola, which leads to more sliding and therefore more decelerating friction forces. (2) The approximation of the Stribeck curve varies, which may lead to differences. (3) The handling of forces at the edge of the plane may be different (compare to Experiment 2).

In the case of our contact model the contact points are moved as displayed in Figure 14 (see Figure 8 also). As  $b_{13}$  constitutes the first edge of the cube to reach the end of the plane, it is the first contact point to be split up and moved in the length and width direction. The movement is limited to the respective dimensions of the cube.

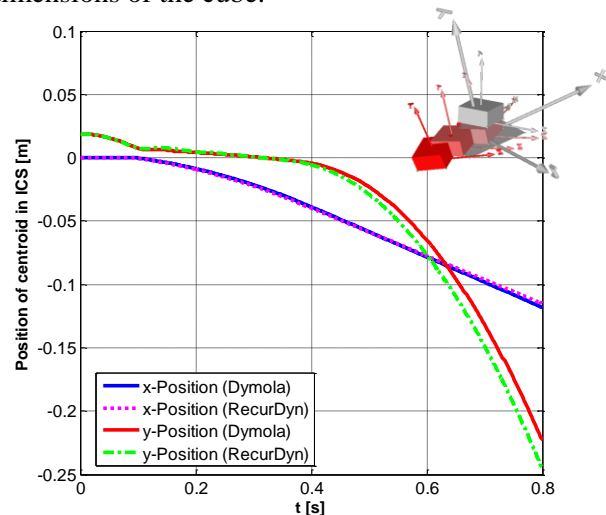


Figure 13: Box falling and sliding on a plane; parameters:  $c = 1e^7$ ,  $d = 1e^7$ ,  $n = 1.1$ ,  $m = 1$ ,  $t = 5e^{-3}$ ,  $\mu_k = 0.08$ ,  $\mu_s = 0.12$ ,  $v_{\varepsilon 1} = 1e^{-4}$ ,  $v_{\varepsilon 2} = 1e^{-2}$ ,  $k_v = 0$

To evaluate the efficiency of the contact library in the context that was outlined in the beginning, we also investigated the ‘‘CPU-time for integration’’. Table 1 compares this characteristic property of the aforementioned models (simulated time  $t = 1s$ ). The results depict the experiences we made in various tests and further experiments. While in general, the integration times are comparable, they strongly depend on the number of contact points in Dymola. One can observe a strong increase, especially when movement of contact points occurs and the calculated displacement is filtered. On the other hand, modeling a rolling body sometimes leads to problems in RecurDyn, which is reflected in the CPU-time (see also Figure 12).

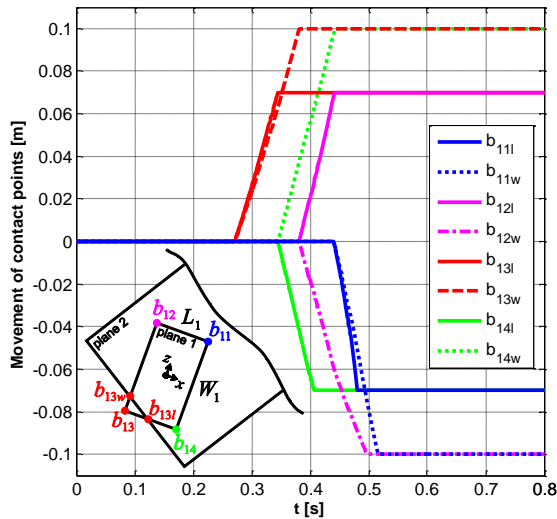


Figure 14: Contact point movement resolved in the local coordinate systems

Table 1: CPU-time for integration<sup>3</sup>

	Dymola	RecurDyn
Experiment 1	0.37s	0.20s (sphere-to-cylinder) 0.50s (solid-to-solid)
Experiment 2	3.52s (filter 10kHz) 0.61s (filter disabled)	4.98s (solid-to-solid)
Experiment 3	5.86s (filter 10kHz) 5.63s (filter disabled)	0.59s (surface-to-surface) 1.90s (solid-to-solid)

## 5 Conclusion and Future Work

In conclusion, we state that our idealized contact library provides a powerful and easy to use opportunity to model contact phenomena of simple contact geometries. Results of various experiments were compared and verified by means of analogous simulations in RecurDyn. The chosen architecture with the implemented contact interface and the configurable contact block matches the idea of reusing composable models. It is therefore especially useful in the conceptual design of mechatronic systems.

In the future, we want to provide more contact surfaces. In addition, we will investigate a possibility to save simulation time by disabling the contact calculation when they are not needed. Despite the validation with RecurDyn, comprehensive measurements are necessary. The major drawback of the force-based approaches is that the spring/damper parameters cannot be obtained directly from the given material properties. These parameters also depend on the surface combination and are usually determined experimentally. To tackle this, we plan to

<sup>3</sup> The simulations were performed on an Intel Core2Duo CPU with 2.53GHz and 4GB RAM

provide ready-to-use parameter sets for often-used combinations. After making our library publicly available, we also hope to identify further opportunities for improvement with the help of the Modelica community.

## Acknowledgements

This work was developed in the project “ENTIME: Entwurfstechnik Intelligente Mechatronik” (Design Methods for Intelligent Mechatronic Systems). The ENTIME project was funded by the state of North Rhine-Westphalia (NRW), Germany, and the EUROPEAN UNION, European Regional Development Fund, “Investing in your future”.

In addition, the authors would like to thank the reviewers for reading the text carefully and giving useful suggestions on where to improve both the paper and the library.

## References

- [1] F. Bauer, J. Gausemeier, D. Köchling, F. Oestersötebier: Approach for an Early Validation of Mechatronic Systems using Idealized Simulation Models within the Conceptual Design. In: *Proceedings of the 5th CIRP Conference on Industrial Product-Service Systems*, Bochum, March 14-15, 2013
- [2] T. Juhász. Advanced Solutions in Object-Oriented Mechatronic Simulation. Ph.D. Thesis, Dept. of Control Engineering and Information Technology, Budapest University of Technology and Economics, 2008
- [3] M. Otter, H. Elmquist, J. Díaz López. Collision Handling for the Modelica MultiBody Library. In: *Proceedings of the 4th International Modelica Conference*, Hamburg, March 7-8, 2005
- [4] G. Hippmann. Modellierung von Kontakten komplex geformter Körper in der Mehrkörperdynamik. Ph.D. Thesis, Vienna University of Technology, 2004
- [5] K. L. Johnson. *Contact Mechanics*. Cambridge University Press, Cambridge, 1985
- [6] J. J. Kalker. *Three-dimensional elastic bodies in rolling contact*. Kluwer Academic Publishers, Norwell, 1990
- [7] W. Sextro. *Dynamical contact problems with friction*. Springer-Verlag, Berlin, Heidelberg, 2007
- [8] D. W. Marhefka, D. E. Orin. A Compliant Contact Model with Nonlinear Damping for Simulation of Robotic Systems. *IEEE*



- Transactions on Systems, Man, and Cybernetics*, 29(6), 1999
- [9] H. M. Lankarani, P. E. Nikravesh. Continuous Contact Force Models for Impact Analysis in Multibody Systems. *Nonlinear Dynamics*, 5, 1994
- [10] K. H. Hunt, F. R. E. Crossley. Coefficient of restitution interpreted as damping in vibroimpact. *ASME J. Appl. Mech*, 1975
- [11] C. Makkar, W. E. Dixon, W. G. Sawyer, G. Hu. A New Continuously Differentiable Friction Model for Control Systems Design. In: *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Monterey CA, July, 2005
- [12] V.L. Popov. *Kontaktmechanik und Reibung*. Springer-Verlag, Berlin, Heidelberg, 2009
- [13] B. Roh, H. Aum, D. Bae, H. Cho, H. Sung. A Relative Contact Formulation for Multibody System Dynamics. *KSME International Journal*, 14(12): 1328-1336, 2000



# The OneWind<sup>®</sup> Modelica Library for Wind Turbine Simulation with Flexible Structure — Modal Reduction Method in Modelica

Philipp Thomas   Xin Gu   Roland Samlaus   Claudio Hillmann   Urs Wihlfahrt  
Fraunhofer Institute for Wind Energy and Energy System Technology IWES  
Am Seedeich 45, 27572 Bremerhaven, Germany

## Abstract

The OneWind<sup>®</sup> Modelica Library<sup>1</sup> [15] for coupled wind turbine loads calculation developed at Fraunhofer IWES uses a structural element based on a modal reduction method to model the motion and deformation of flexible wind turbine rotor blades and tower. The degrees of freedom (DOF) are rigid body motions and modal DOF. The `ModalElement` model allows the simulation of coupling effects like bend-twist coupling in wind turbine rotor blades and the structural behavior is dependent on the selected eigenmodes. This paper gives an overview about the Modelica implementation of the theory of modal elements, the advantages over other methods (finite-elements), how the `ModalElement` model is included into the OneWind<sup>®</sup> Modelica Library, and how it is used for load calculation.

*Keywords: modal, blade, OneWind, OneModelica, load calculation, wind turbine loads, coupled wind turbine simulation*

## 1 Introduction and Motivation

In wind turbine simulations, flexible multibody dynamics is widely used to describe the large motion and flexible deformation of blades. A flexible multibody system is based on the floating Frame of reference formulation. In a standard flexible multibody method, a finite element discretization is often combined with the multibody formulation to describe the beam flexibility. In the finite element method, the flexible beam is discretized into several beam elements. Each element is described with local shape functions weighted with nodal displacements. The numerical accuracy of the finite element method is dependent on the number of elements. A large number of elements demand quite a

lot computational costs, which are usually not suitable for wind turbine simulation.

A modal reduction approach can dramatically reduce the number of DOF of the beam by transforming nodal coordinates to modal coordinates. This approach is based on the assumption that the displacement of the beam can be expressed as the superposition of a series of eigenmodes weighted with modal coordinates.

The objective is to achieve the same accuracy as the finite element method with less computational time. This is a requirement for a wind turbine load calculation with a large number of load cases which are necessary for fatigue analysis. Therefore, the modal reduction method is used to create a structural element in the Modelica programming language which can represent the motion and deformation of flexible components of a wind turbine. With the OneWind<sup>®</sup> Modelica Library, a package of necessary Modelica models for aero-servo-elastic wind turbine simulations is finally available.

The paper is structured as follows: Section 2 describes the theoretical approach of flexible multibody and concludes with the equation of motion in modal coordinates. Section 3 shows the implementation of the flexible multibody with modal reduction in Modelica. In Section 4, an overview of the OneWind<sup>®</sup> Modelica Library with `ModalElement` model for rotor blades and tower is given. Moreover, the internal layout of wind turbine components is shown with a focus on interaction between models. Section 5 shows results of wind turbine deflection with the `ModalElement` model for flexible tower. Lastly, Section 6 concludes the presented work and gives an outlook for further applications of the `ModalElement` model.

<sup>1</sup>Version 1.0 for onshore wind turbine load calculation was released and is now available under dual license model. For further information contact: [info@onewind.de](mailto:info@onewind.de)

## 2 Flexible multibody with modal reduction

The ModalElement model is developed based on multibody dynamics and a modal reduction approach. The multibody system is based on the floating reference Frame. The floating reference Frame indicates that the deformation of the body can be formulated with respect to the corresponding body coordinate system, which moves together with the body. The absolute motion of an arbitrary point  $P$  on the body is defined by

$$r_P = R + A(\bar{u}_0 + \bar{u}_f), \quad (1)$$

where  $R$  represents the location of the origin of the body reference,  $A$  is a transformation matrix,  $\bar{u}_0$  is the local position vector in undeformed state, and  $\bar{u}_f$  is the deformation vector.

The representation of a deformable body in the world Frame requires  $6 + N$  DOF, which includes 6 rigid translational and rotational DOF and  $N$  flexible DOF

$$q = [q_r \quad q_f]^T = [R \quad \theta \quad q_f]^T, \quad (2)$$

where  $\theta$  are the orientation angles of the reference Frame and  $q_f$  is the  $N$ -dimensional vector of flexible DOF. In the Modelica model,  $q_f$  is formulated based on the finite element method and thus,  $q_f$  represents the nodal displacements in the reference Frame.

Using the principle of virtual work [14], the equations of motion for a flexible body in a multibody system can be described as

$$\begin{bmatrix} m_{RR} & m_{R\theta} & m_{Rf} \\ m_{\theta R} & m_{\theta\theta} & m_{\theta f} \\ m_{fR} & m_{f\theta} & m_{ff} \end{bmatrix} \begin{bmatrix} \ddot{R} \\ \ddot{\theta} \\ \dot{q}_f \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & K_{ff} \end{bmatrix} \begin{bmatrix} R \\ \theta \\ q_f \end{bmatrix} + \begin{bmatrix} C_R^T \\ C_\theta^T \\ C_f^T \end{bmatrix} \lambda = \begin{bmatrix} Q_{eR} \\ Q_{e\theta} \\ Q_{ef} \end{bmatrix} + \begin{bmatrix} Q_{vR} \\ Q_{v\theta} \\ Q_{vf} \end{bmatrix}. \quad (3)$$

A modal approach is implemented in the Modelica model to reduce the size of the system. In the modal approach, the specified displacement fields are formulated with the eigenvectors of a series of eigenmodes. The generalized displacement expression can be written as

$$q_f = \sum_{k=1}^n \phi_k Y_k, \quad (4)$$

where  $\phi_k$  is the eigenvector in the  $k^{th}$  mode and  $Y_k$  is the generalized modal coordinate in the  $k^{th}$  mode. The modal coordinate is time dependent, which is solved together with the multibody system. The eigenvector

is calculated in the preprocessing with OneModelica (see Section 4.6).

Equation (4) can be rewritten in the matrix form as

$$q_f = \phi Y, \quad (5)$$

where  $Y$  are the modal coordinates and  $\phi$  is the eigenvector matrix given by

$$\phi = [\phi_1 \quad \phi_2 \quad \dots \quad \phi_n]. \quad (6)$$

This procedure is the modal displacement superposition method. The number  $n$  represents the number of DOF in modal analysis. According to the characteristics of modal analysis, the first several modes dominate the contribution to the displacement field. A proper selection of these modes leads to a good approximation of displacements with a small number of DOF. For example, if  $m$  eigenmodes are selected,  $q_f$  can be described approximately by

$$q_f \approx \phi_m Y_m. \quad (7)$$

Using the eigenvector matrix as the modal transformation matrix, Equation (3) can be written in terms of modal coordinates as

$$\begin{bmatrix} m_{RR} & m_{R\theta} & m_{Rf}\phi_m \\ m_{\theta R} & m_{\theta\theta} & m_{\theta f}\phi_m \\ \phi_m^T m_{fR} & \phi_m^T m_{f\theta} & \phi_m^T m_{ff}\phi_m \end{bmatrix} \begin{bmatrix} \ddot{R} \\ \ddot{\theta} \\ \dot{Y}_m \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \phi_m^T K_{ff} \phi_m \end{bmatrix} \begin{bmatrix} R \\ \theta \\ Y_m \end{bmatrix} + \begin{bmatrix} C_R^T \\ C_\theta^T \\ C_f^T \phi_m \end{bmatrix} \lambda = \begin{bmatrix} Q_{eR} \\ Q_{e\theta} \\ \phi_m^T Q_{ef} \end{bmatrix} + \begin{bmatrix} Q_{vR} \\ Q_{v\theta} \\ \phi_m^T Q_{vf} \end{bmatrix}, \quad (8)$$

where all matrices related to flexible DOF are modal reduced if the number of modal coordinates  $Y_m$  is less than the number of nodal coordinates  $q_f$ .

## 3 Implementation in Modelica

A flexible multibody model with modal reduction has been implemented by using the Modelica language. This model is named ModalElement (see Figure 1) in the OneWind<sup>®</sup> Modelica Library. Frame\_a and Frame\_b are the interfaces which are used to connect to other structural components. Frame\_c is the interface which is used to connect external loads.

The modal shape matrix, the modal reduced stiffness matrix, and the modal reduced mass matrix are

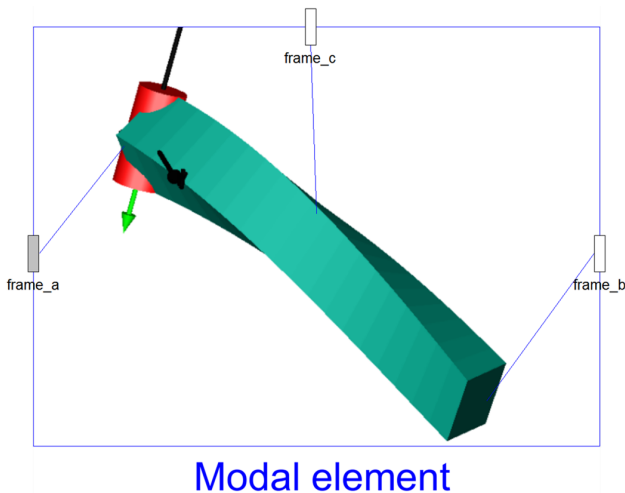


Figure 1: ModalElement icon

calculated in Java in a preprocessing step due to problems with calculating large matrices in Modelica (the details are described in Section 4.6).

The input modal reduced representations of the model are as follows:

```
parameter Real phi[divisions*6,nModes]
=modalElementData.phiMatrix;
parameter Real modalReducedStiffnessMatrix[nModes,nModes]=modalElementData.modalReducedStiffnessMatrix;
parameter Real modalReducedMassMatrix[nModes,nModes]=modalElementData.modalReducedMassMatrix;
```

Listing 1: Modal reduced representations

Dynamic equations (see Equation (8)) of ModalElement with damping are implemented in Modelica language as follows:

```
M*ddQ + D*dQ[7:6+nModes] + K*Q[7:6+nModes]
+ CqT[1:6+nModes,:]*lambda = Qe + Qv;
```

Listing 2: Dynamics equations

### 3.1 Boundary conditions

To solve a finite element problem, suitable boundary conditions are defined in the finite element system. In a pure finite element system, boundary conditions are usually enforced externally. However, in our flexible multibody formulation, boundary conditions are enforced internally in the model. This is because the floating Frame only exchanges the location of the origin of the body reference, the orientation of the body

reference, and the forces with other components. The deformation displacements of the flexible multibody model are not accessible externally.

The boundary condition for ModalElement is the “clamped-free” boundary condition. In this model all six flexible DOF are fixed at the root of the beam. Both displacements and rotations are equal to zero as

$$q_{f1} = q_{f2} = q_{f3} = q_{f4} = q_{f5} = q_{f6} = 0. \quad (9)$$

### 3.2 Animation

Animation is an important feature in wind turbine simulation. The immediate visual feedback about the dynamic behavior of the wind turbine helps the user with troubleshoot during a simulation. The 3D animation of ModalElement is realized by using the Surface function in Dymola with the following code:

```
model ModalElementSurface
extends Modelica.Mechanics.MultiBody.Visualizers.Advanced.Surface(redeclare function surfaceCharacteristic=OneWind.BeamElement.FlexibleMultibody.BeamVisualizer.BeamVisualizer (
    radius=radius,
    extra=1,
    r=r_i,
    qf=qf,
    angle=angle,
    der_angle=der_angle,
    scalingFactor=scalingFactor));
parameter Integer nl "number of visualization points along length";
parameter Real scalingFactor "scaling factor to visualize deformation";
input Modelica.SIunits.Position r_i[nl,3] "initial Position of visualization points";
input Modelica.SIunits.Distance qff[nl,3] "displacements of visualization points";
input Modelica.SIunits.Angle angle[nl,3] "angle at visualization points";
input Modelica.SIunits.AngularVelocity der_angle[nl,3] "angular velocity at visualization points";
end ModalElementSurface;
```

Listing 3: 3D ModalElement animation

## 4 OneWind<sup>®</sup> Modelica Library for coupled flexible wind turbine simulation

The OneWind<sup>®</sup> Modelica Library is based on the MultiBody Library from the Modelica Standard Library and contains all models for aero-servo-elastic load simulation. The models are grouped according to wind turbine components as seen in Figure 2 for an

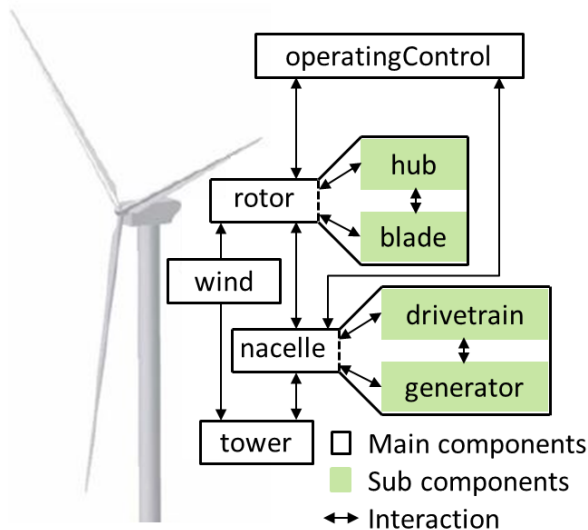


Figure 2: Onshore wind turbine with corresponding component instances and interaction

onshore wind turbine. This Figure shows the model instances and corresponding interaction for the main wind turbine components: the rotor (with the sub-components of blade and hub), the tower, the nacelle (with the sub-components of drivetrain and generator), operating control, and the wind. The corresponding Modelica model for the main wind turbine components is written as:

```

model WindTurbine
  extends OneWind.OnshoreWindTurbine
  (
    //=== rotor ===
    redeclare OneWind.RotorModal rotor
    //=== tower ===
    ,redeclare OneWind.TowerModal tower
    //=== nacelle ===
    ,redeclare OneWind.NacelleRigid nacelle
    //=== operating control ===
    ,redeclare OneWind.Control operatingControl
    //=== wind ===
    ,redeclare OneWind.WindTurbulent wind
  );
end WindTurbine;

```

Listing 4: Modelica representation of Figure 2

To simulate at different levels of detail, several models for almost every wind turbine component are available. To exchange component models, the Modelica concept of replacing classes in the modifier with redeclare (with corresponding replaceable), as seen in Listing 3 and Listing 4, is used. The model OnshoreWindTurbine contains the declaration of instances of necessary wind turbine component models with all connections. These models are

exchanged in WindTurbine while extending from the OnshoreWindTurbine model. To ensure subtype compatibility, models that represent one turbine component extend the same partial model, which contain a replaceable, basic parameter record definition, and the external interface (Figure 3).

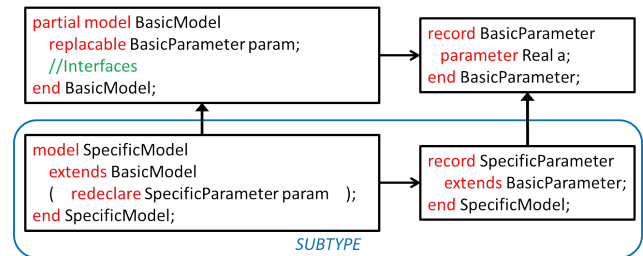


Figure 3: Inheritance concept for subtype compatibility

#### 4.1 Rotor model

The rotor model is the most important component of the wind turbine since it transforms wind velocity through aerodynamic lift to torque, which drives the electrical generator.

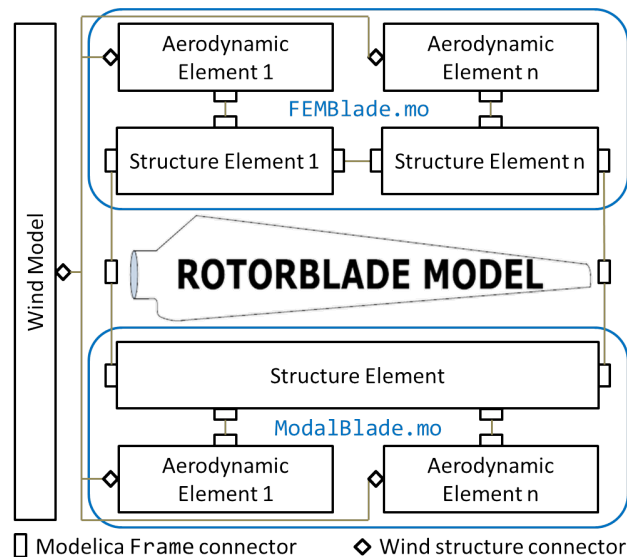


Figure 4: Different layout between the modal and finite element model and interaction between the structure model and the aerodynamic model in the rotor blade model

The rotor blade model consists of two main components which are connected via the Frame connector: the aerodynamic load element and the structure ele-

ment. In Figure 4, the connector scheme and the interaction between the structure model and the aerodynamic model is shown. The rotor blade models differ in regards to their structural part. The `ModalBlade` model consists of only one structure element, the `ModalElement` model, whereas the `FEMBlade` modal consists of  $n$  structure elements (in case of the NREL 5-MW reference wind turbine [7],  $n$  is 17 per blade and 11 for the tower). The structure elements are internally connected to the blade root, the blade tip, and to the aerodynamic load element through the use of the `Frame` connector. All rotor blade models extend a `PartialBlade` model, where the aerodynamic load element model is defined as well as the external interface root `Frame`, tip `Frame`, and the specific connector `StructureWind`, which is used to connect the wind velocity and position to the aerodynamic load element.

#### 4.1.1 Wind models

Deterministic and stochastic wind models are available. Deterministic wind uses several gust models. The stochastic wind data is read from binary or ASCII file in TurbSim [8] format.

All wind models consider the effect of the tower on wind velocity: in case of upwind, the potential flow method is applied and in case of downwind, an empirical model is used. Furthermore, exponential or logarithmic wind shear and vertical and horizontal inclination angle are accounted. The wind velocity is given as an input to the `StructureWind` connector.

All available wind models fulfill requirements defined in IEC 61400-3 [5].

#### 4.1.2 Aerodynamic models

Inside the aerodynamic load element, the `StructureWind` connector delivers the wind velocity which is used to calculate aerodynamic forces and moments. The aerodynamic load calculation uses either the blade element momentum theory (BEM, described in [11], [2], and extended in [1], [3]) or the general dynamic wake (GDW, described in [4], [16]) method. The BEM is implemented as an iterative algorithm while the GDW is a set of differential equations. For the BEM, corrections for the dynamic wake and the dynamic stall can be used. Dynamic stall correction is also available for GDW. The aerodynamic forces and moments are given as an input to aero load `Frame` connector.

#### 4.1.3 Blade models

Blade models differ in regards to their representation of flexible body motion. The `ModalBlade` model uses modal degrees of freedom, the `FEMBlade` model based on the Bernoulli beam element formulation uses 6 degrees of freedom at each node, and the `RigidBlade` model has no flexibility at all (only rigid body motion). The flexible blades consider centrifugal stiffening and pre-twist. All formulations are linear which are the standard in wind turbine load calculations. Non-linear formulations for large deflections but small deformations are under development.

Aerodynamic loads from the aero load `Frame` connector act as external forces and moments and lead to the rotation of the blades. The resulting forces and moments are transferred to the blade root `Frame` connector. Aero-Structure-Coupling enables the simulation of the coupling between the rotor blade deflection and the aerodynamic load calculation.

#### 4.2 Tower models

As for blade models, rigid and flexible models are available for the tower. The structure is basically the same: on the outside, the tower consists of a bottom `Frame`, a tip `Frame`, and load `Frame` connectors along its vertical length. Internally, the tower consists of the structure elements for load and deflection calculation and load elements for the calculation of the aerodynamic drag force from the wind. The connections are realized via `Frame` connectors (except for wind which uses a specific connector similar to the rotor model).

#### 4.3 Control system

The operating control system is implemented as discrete algorithms based on [7] and consists of a PI-pitch algorithm to control the power production above rated rotor rotation speed and a generator-torque controller which requests the counter torque from the electrical generator due to rotor torque. A single pole, low pass speed filter with cut-off frequency of 0.25 Hz transfers the continuous generator rotation speed signal to a discrete signal for control logic. For communication of control signals between sensors inside the wind turbine components and operating control, the `SignalBus` and the `SignalSubBus` connectors are used to build a hierarchic network of expandable connectors (as described in [9]).

#### 4.4 Offshore wind turbine components

Several physical models in the Modelica language for the dynamics of offshore wind turbines and the modeling of water and waves are under development. Fraunhofer IWES is a participant of IEA Wind and the Cooperative Agreement<sup>2</sup> with OC3 [17] and OC4 [10] projects to compare offshore wind turbine simulation codes, where the developed models for offshore wind turbine simulation are verified (see [18]).

#### 4.5 Interfaces between component models

The OneWind<sup>®</sup> Modelica Library basically uses three types of interfaces:

- Modelica Frame connector,
- Modelica FFlange connector,
- Component specific connector.

The Frame connector is utilized whenever loads are exchanged between different models and so does the aerodynamic load element calculate forces and moments from wind velocity using an aerodynamic theory. The transfer between aerodynamic load element and rotor blade is done via a Frame connector.

The FFlange connector is used for a limited degree of freedom representation of the drivetrain dynamics and transfers the torque and angle from the rotor to the electrical generator.

Additionally, user-defined connectors are used to transfer specific information between models that are not covered by the Frame or FFlange connector. Figure 4 exemplarily shows a user defined connector that transfers the wind velocity and position along the structure components to the corresponding aerodynamic load element.

#### 4.6 OneWind<sup>®</sup> Modelica Library with OneModelica

The Modelica IDE OneModelica [12], which is based on the Eclipse RCP framework, is being developed at Fraunhofer IWES. Figure 5 illustrates the workflow of the Modelica model development with OneModelica.

Modelica models can be developed by reusing models from libraries like the Modelica Standard Library and the OneWind<sup>®</sup> Modelica Library. Both libraries are integrated into OneModelica by default. Additionally, OneModelica also provides functionality for the generation of turbulent wind (according to Kaimal [6]). Input files for Modelica models are generated in

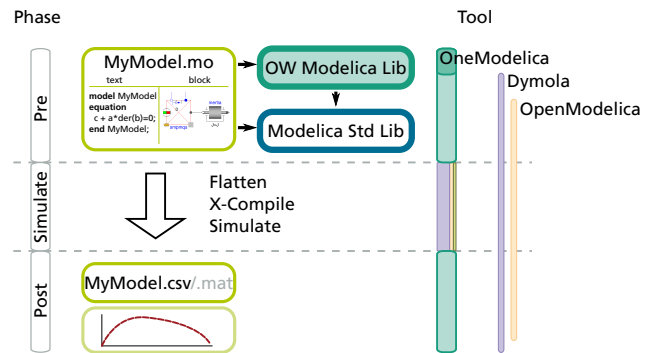


Figure 5: Modelica workflow with OneModelica

binary or ASCII format. The transformation of structure components into model reduced representations has been implemented in Java. This was necessary due to problems with large matrices when calculating the eigenfrequencies directly in Modelica.

In OneModelica, input models for the transformation are automatically recognized and a marker is added to the model files. Input models need to be of type record and must extend the record ModalInput which is included in the OneWind<sup>®</sup> Modelica Library. By right-clicking on the model file, the user can open the transformation dialog and provide a path and file name for the generated modal representation.

The record ModalInput defines distributed structural properties. At the user defined record level, parameters have to be redefined and the needed information for the FEM structure model have to be provided. In this way, the usage of the same data for modal and finite element model is ensured. The parameters are then extracted from the Modelica file and used by the transformation code developed with Java. The generated Modelica record with modal data contains the modal reduced mass and the stiffness matrix as well as the eigenforms and can be used with modal components. Changes in the structure model can easily be transferred to the modal component by the regeneration of the modal record.

The same workflow is used for the generation of turbulent wind data. A template record must be extended and parameterized. The generated wind file can then be used by a Modelica wind field generator that is included in the OneWind<sup>®</sup> Modelica Library. The user can re-use the generator and provide a path to the generated wind file.

Besides the described functionality, OneModelica supports several Modelica simulation tools for doing the flattening, compiling, and simulation and several simulations can be run in parallel. After a simula-

<sup>2</sup>[http://www.ieawind.org/about\\_co-operative\\_agree.html](http://www.ieawind.org/about_co-operative_agree.html)



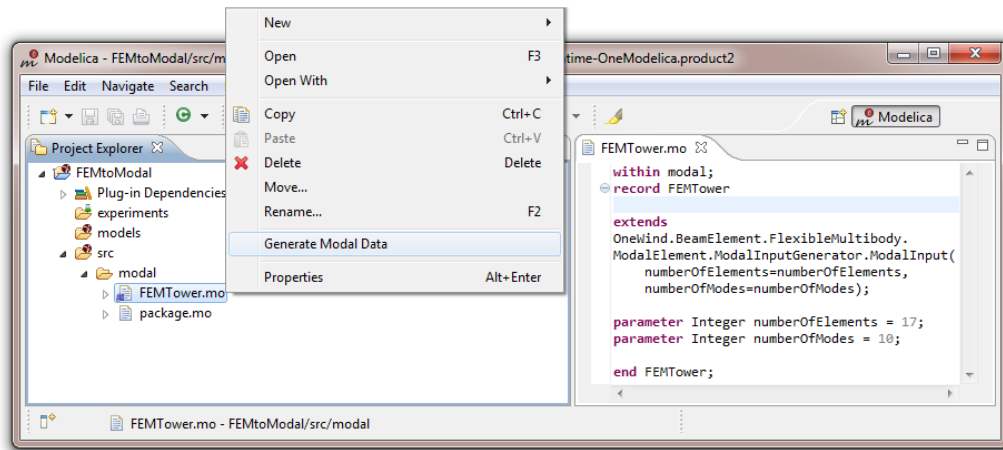


Figure 6: Transforming structure models to modal format

tion, the results are transferred back to OneModelica for post processing. Simulation settings and results are stored. The latest enhancement in OneModelica is an integrated framework for the automatic Modelica model testing [13].

## 5 Simulation Results

In the following section, simulation results with flexible wind turbine components are presented. All mentioned results were computed in Dymola<sup>3</sup>.

Figure 7 compares the deflection obtained with the ModalTower model and the finite element based TubularTowerModel model at the tower top Frame connector in the x- (downwind) and y-direction (side-to-side). The tower model is parametrized according to the NREL 5-MW reference wind turbine with the addition of a constant three dimensional force at the tower top. Both flexible models oscillate at the same frequency at each respective direction. The ModalTower model results strongly depend on the number of modes. The selected modes from the modal reduction must be able to represent the motion, which are enforced to the structure by external loading. Table 1 shows the error for a selection of eigenmodes (for a detailed view on convergence of error up to 30 Modes, see Figure 8). The reference is always the finite element model. Since the first eigenmode is only a 1-directional bending mode, it can only represent the bending in x-direction. Using more than 6 modes causes the error to drop below 1% in both directions. This means that 6 flexible (modal) DOF are enough

to represent the deflection of the ModalTower with almost the same accuracy as with TubularTowerModel. A reduced number of DOF will result in less computational time, especially with coupled wind turbine simulation.

To compare CPU-time, the load simulation of an aero-servo-elastic wind turbine is utilized. The model is parametrized according to the NREL 5-MW reference wind turbine. For the tower, the ModalTower model with six eigenmodes or the finite element based TubularTowerModel model is used. Integration algorithms are DASSL and Euler mixed<sup>4</sup>. For ModalElement based models, it is possible to choose Rkfix4<sup>5</sup>, which is not possible for the finite element model where no solution is found. Table 2 contains the results. As mentioned before, a reduced number in flexible DOF result in a reduction of computational time. The increase in simulation performance depends on the integration algorithm and lies between 300% and 550% for Euler mixed and DASSL. With Rkfix4, the simulation reaches real-time abilities.

## 6 Conclusion

This paper describes the approach and the implementation of a flexible structure element based on a modal reduction method in the Modelica programming language and its integration into the OneWind<sup>®</sup> Modelica Library for aero-servo-elastic load simulations of wind turbines.

The ModalElement model delivers comparable structural behavior as the finite element model. The

<sup>3</sup>Dymola 2014 FD01, <http://www.dymola.com>

<sup>4</sup>Mixed explicit/implicit Euler algorithm

<sup>5</sup>Runge-Kutta 4th order integration algorithm

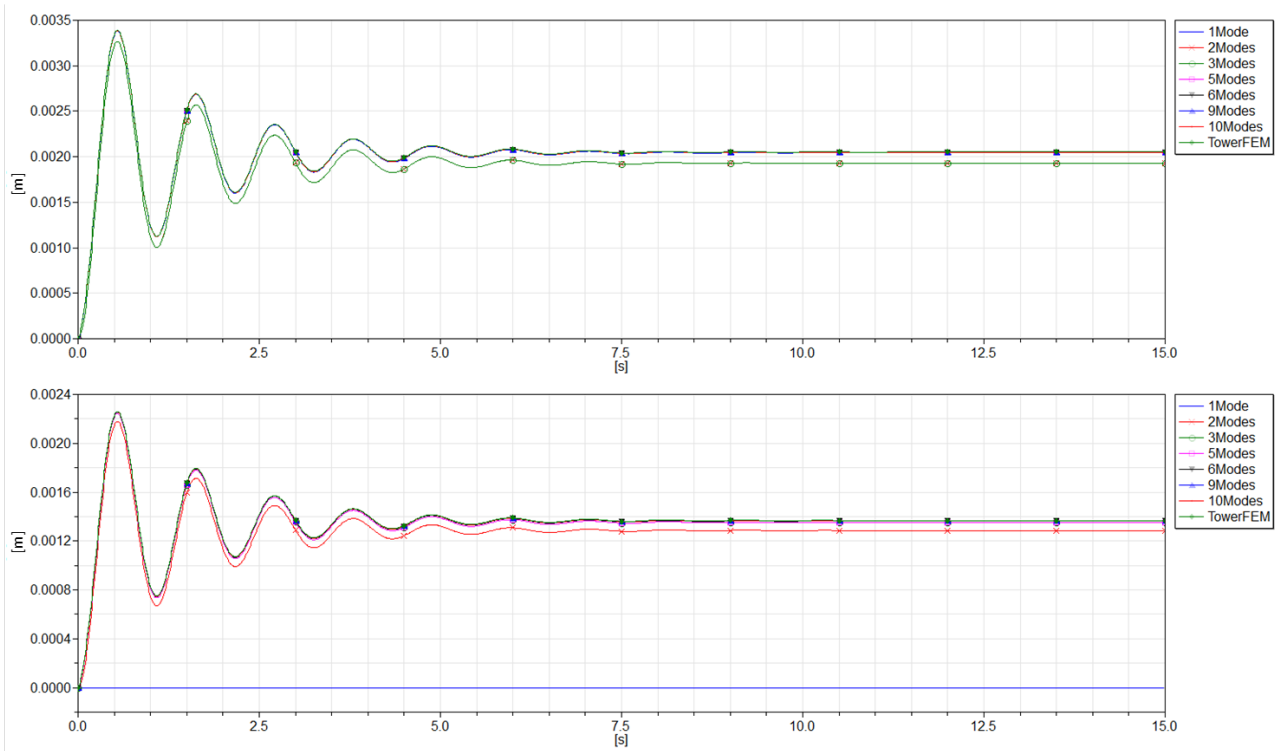


Figure 7: Tower top deflection: upper plot shows downwind displacement, lower plot shows side-to-side displacement

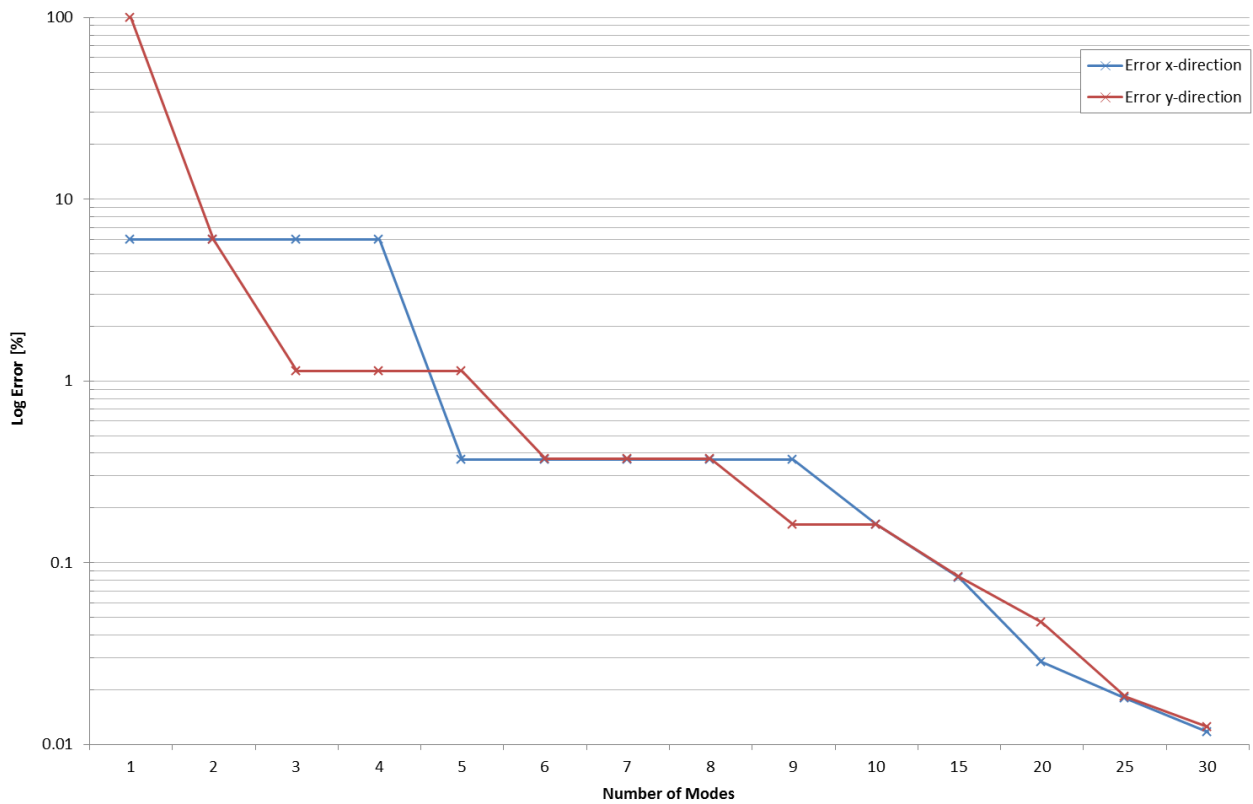


Figure 8: Simulation error for different numbers of Eigenmodes

Table 1: Simulation error for different number of eigenmodes; range: 1 mode (1-10) and 5 modes (10-30)

No. of Modes	x-Deflection [m]	Error [%]
1	1.9292E-03	5.989
⋮	⋮	⋮
5	2.0445E-03	0.371
⋮	⋮	⋮
10	2.0488E-03	0.161
15	2.0504E-03	0.083
20	2.0516E-03	0.028
25	2.0518E-03	0.018
30	2.0519E-03	0.012
FEM	2.0521E-03	-

No. of Modes	y-Deflection [m]	Error [%]
1	8.1974E-14	100.0
2	1.2862E-03	5.9901
3	1.3526E-03	1.1315
⋮	⋮	⋮
6	1.3630E-03	0.3713
⋮	⋮	⋮
9	1.3659E-03	0.1615
10	1.3659E-03	0.1615
15	1.3670E-03	0.0833
20	1.3675E-03	0.0468
25	1.3679E-03	0.0183
30	1.3679E-03	0.0124
FEM	2.0521E-03	-

Table 2: CPU-time for 10 s simulations of a wind turbine with a flexible tower

Algorithm	FEM	Modal	Speed-up
Euler mixed	187 s	61 s	307%
DASSL	2230 s	400 s	557.5%
Rkfix4	-	8.6 s	-

models oscillate with the same frequency. The error in deflection calculation drops below 1% beginning from the use of 6 eigenmodes. At this configuration, the simulation results are almost the same as with the finite element model and are available at a fraction of the finite element model computational time (see Table 2). Therefore, the ModalElement model is applicable for the load simulations of a flexible wind turbine.

The next step will be the application of the ModalElement model on substructure components for offshore wind turbine simulations and the integration to the OneWind<sup>®</sup> Modelica Library together with models for water and waves. Further development of the OneWind<sup>®</sup> Modelica Library aims to add:

- A structure element model for large deflections and large rotations,
- A state-machine based supervisory control,
- An interface for Bladed-style DLL for the controller,
- Real-time capability for a coupled flexible wind turbine simulation.

## Acknowledgements

This work is financially supported by the Federal Ministry for the Environment, Nature Conservation, and Nuclear Safety based on a decision by the Parliament of the Federal Republic of Germany.

## References

- [1] Betz A. Das Maximum der theoretisch möglichen Ausnutzung des Windes durch Windmotoren. In: Zeitschrift für das gesamte Turbinenwesen, 26:307-309, 1920.
- [2] Froude R.E. On the part played in propulsion by differences of fluid pressure, translations of the Institution of Naval Architects, 30:390-405, 1889.
- [3] Glauert H., Division L. Airplane propellers, aerodynamic theory, volume 4, Durand WF, Berlin, Germany, 1935.
- [4] He C. Development and Application of a Generalized Dynamic Wake Theory for Lifting Rotors, PhD thesis, Georgia Institute of Technology, 1989.
- [5] IEC. Wind turbines – Part 3: Design requirements for offshore wind turbines, IEC 61400-3, 1.0 edition, 2009.

- [6] Izumi Y., Kaimal J.C., Wyngaard J.C., Cote O.R. Spectral characteristics of surface-layer turbulence. In: Q.J.R. Meteorol. Soc., volume 98, pp. 563-598, 1972.
- [7] Jonkman J., Butterfield S., Musial W., Scott G. Definition of a 5-MW reference wind turbine for offshore system development, Technical Report NREL/TP-500-38060, National Renewable Energy Laboratory (NREL), Golden, Colorado, USA, 2009.
- [8] Jonkman J., Kilcher L. TurbSim user's guide: version 1.06.00, National Renewable Energy Laboratory (NREL), Golden, Colorado, USA, 2012.
- [9] Otter M. Modeling, simulation and control with Modelica 3.0 and Dymola 7, technical report, Deutsches Zentrum für Luft- und Raumfahrt e.V. DLR - Institut für Robotik und Mechatronik, Wessling, Germany, 2009.
- [10] Popko W., Vorpahl F. Offshore Code Comparison Collaboration (OC4) Project – Task 30. In: IEA Wind 2012 Annual Report, Chapter 10, 45-48, PWT Communications, July 2013.
- [11] Rankine W.J. On the mechanical principles of the action of propellers, translations of the Institution of Naval Architects, 6:13-30, 1865.
- [12] Samlaus R, Hillmann C, Demuth B, Krebs M. Towards a model driven Modelica IDE. In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany, Modelica Association, 20-22 March 2011.
- [13] Samlaus R., Strach M., Hillmann C., Fritzson P. MoUnit – A framework for automatic Modelica model testing, 10th International Modelica Conference 2014, Lund, Schweden, 2014.
- [14] Shabana, Ahmed A. Dynamics of multibody systems, Cambridge University Press, New York, USA, 2005.
- [15] Strobel M., Vorpahl R., Hillmann C., Gu X., Zuga A., Wihlfahrt U. The OnWind Modelica Library for offshore wind turbines – Implementation and first results. In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany, Modelica Association, pp. 603 - 609, 20-22 March 2011.
- [16] Suzuki, A. Application of Dynamic Inflow Theory to Wind Turbine Rotors, PhD thesis, The University of Utah, 2000.
- [17] Vorpahl F., Strobel M., Jonkman J., Larsen T., Passon P. and Nichols J. Verification of aeroelastic offshore wind turbine design codes under IEA Wind Task XXIII. Wind Energy, doi: 10.1002/we.1588.
- [18] Wojciech P., Vorpahl F. et al. Offshore Code Comparison Collaboration Continuation (OC4), Phase I – Results of coupled simulations of an offshore wind turbine with jacket support structure. In: Proceedings of the 22nd International Offshore and Polar Engineering Conference (ISOPE), Rhodes, Greece, International Society of Offshore and Polar Engineers, pp. 337-346, June 2012.

# Simulating Collisions within the Modelica MultiBody Library

Andreas Hofmann<sup>1</sup> Lars Mikelsons<sup>1</sup> Ines Gubsch<sup>2</sup> Christian Schubert<sup>2</sup>

<sup>1</sup>Bosch Rexroth AG, Lohr am Main, Germany

<sup>2</sup>TU Dresden, Chair of Construction Machines and Conveying Technology, Germany

## Abstract

In this paper an approach for handling collision within the Modelica MultiBody library is presented. Therefore, a short overview about collision consideration for multibody simulation is given. Different methods for calculating the contact reactions are discussed and their potentials for implementation in a free Modelica library are deliberated. Furthermore the implementation of this collision library, using a penalty-based collision approach and the *Bullet Physics Library* for collision detection is described. The application is demonstrated in examples and limitations are brought up. Although some drawbacks restrict usability, the library can be used to increase the level of detail for multibody simulation models.

*Keywords:* contact; collision detection; collision library

## 1 Introduction

### 1.1 Motivation

Many physical systems cannot be simulated in a feasible manner without the description of collision interaction. Not only the typical applications, like wheel-road-contact, newton's cradle or a bouncing ball need collision consideration, but especially real-life models require contact handling. For example, simulation of typical working processes for construction machines, with lifting rocks can benefit from this. But also machine elements like mechanical springs require collision handling for simulations including dynamic loads.

For the Modelica Library *Modelica.Mechanics.MultiBody* (*M.MB*) several collision handling considerations have been made, with two to be shortly mentioned. In [1] Otter et al. introduced an extension to the *M.MB* library with capabilities of handling collisions. Collision detection using different approaches of surface representation were

shown. Engelson [2] described a way of contact implementation using impulse-based and penalty-based methods. However, those approaches have never been available in public.

To offer collision handling to general public, *CollisionLib* – the library presented here – will be freely available. Although the functionality of this very first version has only been tested in Dymola, support for OpenModelica and other Modelica environments are planned for the future.

### 1.2 Outline

In the following section general information about collision handling is given. The main steps for treating contacts are considered and several methods for handling collisions are described. These methods are compared with respect to their capabilities of straightforward implementation in Modelica.

The next chapter addresses *CollisionLib* - a library for collision handling within Modelica. A summary about requirements and intentions is given and, moreover, the implementation of collision detection and collision response is described.

Examples and limitations of applications are shown in section 4 and the paper is closed with an outlook about further development.

## 2 Collision handling approaches

### 2.1 Main aspects of collision handling

When handling collision interaction in multibody simulations two main steps have to be considered.

(1) **Collision detection** needs to be performed for every possible contact pair of bodies. By testing each combination the effort  $e_T$  is  $O(n^2)$ , see (1), where  $n$  is the total number of bodies.

$$e_T = \frac{n \cdot (n - 1)}{2} = \frac{n^2 - n}{2} \quad (1)$$

It follows from the foregoing that collision detection might have an impact on simulation time.

In order to accelerate this process different two step methods have been developed. Firstly, in the so called broadphase, a rough estimation which bodies might collide is made. Hence, all body combinations that cannot interact for obvious reasons are rejected. Those couples, that might collide, are tested with a proper distance algorithm, e.g. GJK [11], in the second step, called narrowphase.

By doing so, the overall effort can be reduced considerably. In [3] Baraff describes a broadphase method with  $O(n)$ . The total resultant effort  $e_T$  then is  $O(n+k)$ , with  $k$  being the number of body pairs requiring proper examination.

In addition to finding colliding body pairs, the corresponding contact normal and tangent vectors have to be determined for each of these pairs. For some approaches even more data has to be provided.

**(2) Collision reactions** have to be computed according to the chosen collision approach. Since collision handling has a long history, going back to Newton, Poisson, Coulomb and their laws and hypotheses, many different approaches for calculating collision responses are available. However, regarding to Mirtich [4] the state-of-the-art methods can be classified into three categories.

**(a) The Penalty-based approach** is the only one of these three allowing intersection among the bodies. The basic idea is that between the colliding bodies a spring, spring-damper or some other force element is present, which generates separating forces. The forces are related to the penetration of the bodies. This rather simple approach has some disadvantages. Finding right contact parameters is an open problem. Moreover, this parameters cannot be adopted from one simulation to others. Also collision forces have to be big in order to avoid deep intersection between the bodies. This leads to stiff DAE systems which are hard to solve, require small time steps and thus may lead to long simulation times.

**(b) Analytical solutions** prohibit intersection among the colliding partners. The constraints between the involved bodies are conveyed into a linear complementary problem (LCP), see [5] or [6]. Without consideration of friction the LCP formulation will always lead to contact forces, creating realistic movements of the bodies. However, these contact forces do

physically not need to present the right solution since the LCP might find unlimited numbers of solutions. In Figure 1 three possible solutions for the symmetric table under weight force  $w$  are presented with only (c) showing correct behavior. If friction is included

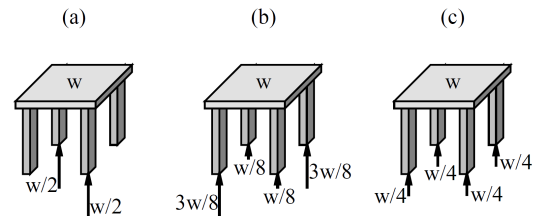


Figure 1: Possible solutions for the a standing table simulation using the LCP approach [4]

into the collision consideration the analytical approach might lead to no or no unique solution. In addition, solving an LCP can become difficult and system information is required, e.g. mass matrix and Jacobian matrix.

**(c) Impulse-based methods** are the third group of approaches for handling collision reactions. All kinds of constraints are neglected and no constraint forces are calculated. Instead all movements are handled using impulses and correcting impulses. Basics of this rather young idea of simulating movements are described in [7], [4]. The impulse-based methods, like the analytical approaches, prohibit intersection between the bodies. But, in contrast to the solutions of the analytical approach, the contact reactions are always calculated physically correct. Of course, also the impulse-based methods have some drawbacks. Stationary contact, like resting bodies on the ground, need to be solved by a high frequent number of small impulses. Interaction between multiple bodies, like a stack of boxes, can cause problems and corrupt results, cf. [4]. The biggest problem, when working with impulse based methods is, that a special form for the equations of motion is needed and special routines for solving those are required.

## 2.2 Implementation in Modelica

Because impulse based methods do not use constraint forces, an implementation in Modelica appears expensive – especially since correction impulses need to be calculated in one simulation step and have to be applied in the previous one. Also the interaction with other physical domains seems difficult.

The analytical approach is more applicable since constraint forces are calculated. However, solving the LCP can be difficult and requires knowledge of the mechanical system, like mass matrix of the mechanical system, etc.. Since Modelica is a multi-domain modeling language collecting the equations from all domains, a straightforward implementation in Modelica is, regarding to the author, not possible.

Hence, for this implementation a penalty method is chosen. For the first part of the collision handling, collisions have to be identified, as described above. Different free collision detection software packages, e.g. *Bullet* or *ODE*, are available, calculating the needed data for a penalty approach. Therefore, integration can be done directly without rewriting any contact routine, etc.. However, drawbacks of the penalty based methods may not be ignored and limitations must be accepted.

### 3 CollisionLib - a contact library for the multibody environment in Modelica

#### 3.1 General aspects

CollisionLib is a library that extends the *Mechanics.MultiBody*-library (*M.MB*) by collision consideration. This expansion implies that existing models do not need to be rebuilt. Instead, the mechanical components are connected to special collision objects, as seen in figure 2.

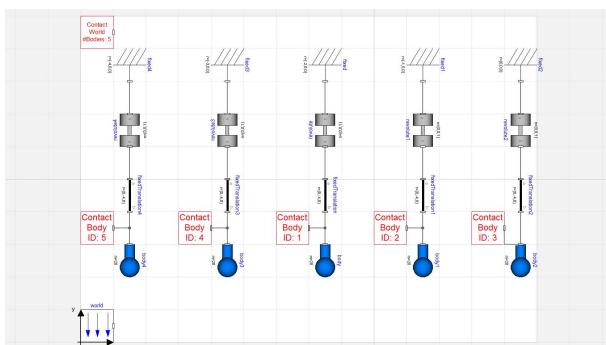


Figure 2: Setup of a mechanical System including collision handling

#### 3.2 Collision detection within Modelica

For solving the issue of collision detection *Bullet Physics Library* (*Bullet*) [8], an external C++-library, is used. It is a free package for simulating mechanical

systems, originating from entertainment industry with focus on video games and movies. Since *Bullet* uses a modular conception, see Figure 3, it is possible to take its collision detection only and drop the simulation routines.

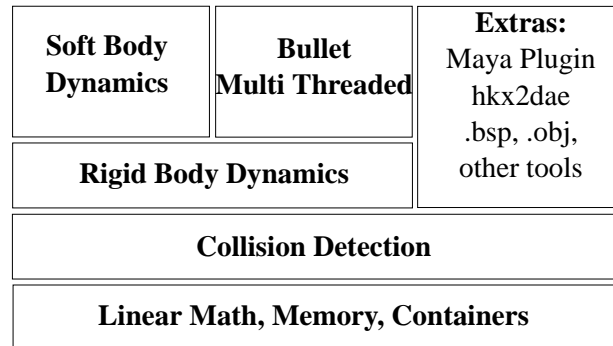


Figure 3: The modular concept of the *Bullet Physics Library* [9]

In order to calculate distance between two bodies, their geometry and their position as well as their orientation must be defined. Within the *M.MB* bodies are represented by their mass and inertia and therefore no additional information is present. However, *Bullet* contains a database with the basic geometries – sphere, box, cone, cylinder and capsule –, able to create a representation, if the right information is passed to it. For complex structures surface data, using a polygon mesh, need to be passed. In collaboration of *Bullet* and Modelica this implies that for basic bodies only some identifier and dimensions have to be supplied. The use of complex, mesh based geometries are currently not included, but planned for future versions.

Due to the fact, that all bodies of the *M.MB* are rigid, meaning that their inertia respectively shape will not change by cause of loads, the geometry information is only passed at initialization of the simulation.

In contrast to shape, the pose of each body needs to be updated every simulation step. This is done by transferring position and rotation matrix to *Bullet* each step.

With all the information described above *Bullet* can perform the check for collisions. From the several available broadphase algorithms within the C++- library an approach using axis aligned boundary box trees (AABB-trees) is used. Detailed information about AABB-trees can be found for example in [12]. For narrowphase intersection tests *Bullet* provides different algorithms for primitive shapes, triangle meshes, etc..

The coupling between Modelica and *Bullet* is done

by expanding the Modelica class *externalObject* into a class called *CollisionWorld* and developing a *Bullet* simulation runtime. Attentive readers might have noticed that *Bullet* is a C++-library and therefore cannot directly be connected to Modelica. To attach the *Bullet Physics Library* to Modelica an additional C-interface is needed. At instantiation of class *CollisionWorld* within a Modelica model, geometry information is passed to the interface. This interface itself, creates an instance of the *Bullet* simulation runtime, passing the position and rotation matrix, and returning a reference to Modelica. This procedure is visualized in Figure 4.

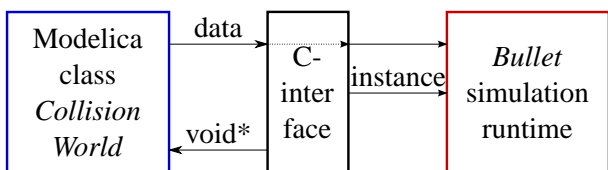


Figure 4: Coupling between Modelica and *Bullet* at system initialization

During each simulation step, the pose of the bodies needs to be updated and collision detection must be performed within the *Bullet* runtime. In Modelica, the function *updatePosition*, of type *externalC*, passes the reference to the *bullet* environment and the pose of the bodies back to the C-interface. This results in an updated *Bullet* simulation, giving back the number of contact body pairs (NoC). Using a second *externalC*-function, named *getCollisionData*, the collision data (coll. data) is given back to Modelica. The complete data flow between Modelica, C-interface and *Bullet* is shown in Figure 5.

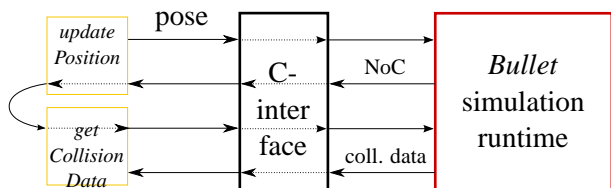


Figure 5: Data flow during simulation

### 3.3 Collision normal force

As specified before, a penalty based approach for calculation collision reactions is used. The approach included in the *CollisionLib* calculates reaction forces using deepest point penetration. This means the two deepest points between the intersection bodies are calculated and the normal vector based on this two is

given by *Bullet*. Another possible penalty method for calculating response, could use consideration of intersecting volume. However, this is not possible using *Bullet*.

There are many different approaches calculating contact normal forces taking geometry, material properties, velocities, mass, etc. into account. A survey about some ideas is given by Machado et al. in [10]. In order to embrace the dimension of methods user-defined collision response laws can easily be implemented.

The basic normal force calculation, available in this library, uses a parallel set of non-linear spring and linear damper. The non-linear spring has its origin in the Hertzian contact theory but is here extended using energy dissipation.

$$F_N = F_c + F_d \tag{2}$$

The part  $F_c$  represents contact force due to the spring and  $F_d$  is the force due to the damper.

$$F_c = K \cdot \delta^n \tag{3}$$

The parameter  $K$  delineates the contact stiffness while the exponent  $n$  describes the non-linear force behavior due to penetration  $\delta$ .

For the damping force some limitations apply. Its magnitude can never exceed the spring part, preventing sticking between the two bodies. Therefore a dummy item  $F_{d2}$  is introduced

$$F_{d2} = D \cdot \dot{\delta} \tag{4}$$

$$F_d = \begin{cases} F_c & \text{if } F_{d2} > F_c \\ -F_c & \text{if } F_{d2} < -F_c \\ F_{d2} & \text{else} \end{cases} \tag{5}$$

The factor  $D$  describes the contact damping and  $\dot{\delta}$  denotes the relative velocity of the contact points.

All parameters ( $K$ ,  $n$ ,  $D$ ) depend on geometry and material of the colliding bodies. For special combinations of bodies different analytical and empirical approaches have been published. The parameters can also be derived by Finite-Element-Analysis or practical test execution. However there are no generalities available.

As mentioned before the contact parameters are individual for each contact. Nevertheless the parameters are globally set for all contacts in the first place. But of course the contact parameters can be individually assigned for each pair of bodies.



### 3.4 Friction force calculation

In order to calculate friction forces it is necessary to determine two tangent vectors ( $\mathbf{tv}_1$ ,  $\mathbf{tv}_2$ ) from the contact normal vector  $\mathbf{nv}$ . Since the two tangent vectors and the normal vector have to be orthogonal to each other two components of one tangent vector can be chosen freely. In *CollisionLib* this is done by the following scheme:

- Determination of the biggest component of the normal vector  $\mathbf{nv}$
- Assignment of value 1 to the two components of the first tangent vector  $\mathbf{tv}_1$ , respectively to the two smaller components of  $\mathbf{nv}$
- Calculation of the last component of  $\mathbf{tv}_1$  using dot product ( $\mathbf{nv} \cdot \mathbf{tv}_1 = 0$ )
- $\mathbf{tv}_2$  is calculated via cross product ( $\mathbf{nv} \times \mathbf{tv}_1$ )
- Normalization of both tangent vectors

With this tangent vectors the tangential plane of the contact is determined, in which the friction force vector is located. In order to identify its direction the projection of the relative velocity vector  $\mathbf{v}_{rel}$  into the tangential plane is required. The vector  $\mathbf{v}_{rel}$  is calculated from the contact pair of deepest penetrating points between the two bodies  $\mathbf{K}_a$  and  $\mathbf{K}_b$  as described from (6) to (8). All calculations are performed with respect to the initial frame  $\{\mathbf{O}^I, \mathbf{e}^I\}$  and refer to Figure 6. To keep calculations as short as possible only the quantities of *bodyB* are derived. The values of *bodyA* are calculated analogical.

First of all, the vector from the body fixed frame  $\{\mathbf{O}^B, \mathbf{e}^B\}$  to the contact point  $\mathbf{K}_b$  is calculated.

$$\mathbf{r}_b = \mathbf{r}_{Kb} - \mathbf{r}_{b0} \quad (6)$$

Using this vector the velocity of the contact point can be obtained.

$$\mathbf{v}_{Kb} = \frac{d\mathbf{r}_{Kb}}{dt} = \dot{\mathbf{r}}_{b0} + \boldsymbol{\omega}_b \times \mathbf{r}_b \quad (7)$$

After determination of  $\mathbf{v}_{Kb}$  the relative velocity is computed:

$$\mathbf{v}_{rel} = \mathbf{v}_{Kb} - \mathbf{v}_{Ka} \quad (8)$$

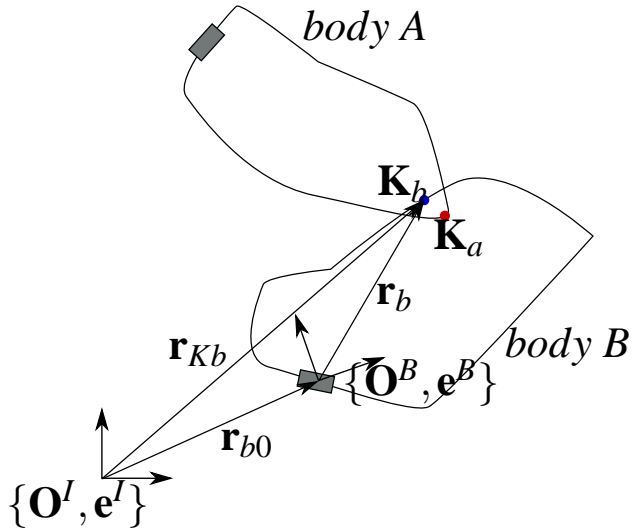


Figure 6: Calculation of the relative velocity during contact

By projection this vector into the tangent plane ( $\mathbf{v}_{relP}$ ) the opposite direction of the friction force vector is derived.

$$\mathbf{v}_{relP} = (\mathbf{tv}_1^T \cdot \mathbf{v}_{rel}) \cdot \mathbf{tv}_1 + (\mathbf{tv}_2^T \cdot \mathbf{v}_{rel}) \cdot \mathbf{tv}_2 \quad (9)$$

The magnitude of  $\mathbf{v}_{relP}$  is compared to a limiting velocity  $v_G$  at which the coefficient of static friction  $\mu_H$  is no longer used and sliding friction  $\mu_G$  is applied.

$$\mu = \begin{cases} \mu_H & \text{if } |\mathbf{v}_{relP}| < v_G \\ \mu_G & \text{if } |\mathbf{v}_{relP}| \geq v_G \end{cases} \quad (10)$$

Along with the magnitude of the contact normal force  $F_N$  the magnitude of the Friction force  $F_{Fmag}$  is calculated.

$$F_{Fmag} = \mu \cdot F_N \quad (11)$$

However, since the contact normal forces can be enormous a user-defined quantity  $F_{Fmax}$  is introduced, allowing to reduce the maximum assignable friction force if wanted, see (12).

$$F_{Fmag} = \min(F_{Fmax}, \mu \cdot F_N) \quad (12)$$

Using the negative normalized vector of the projected relative velocity the friction force vector can be computed.

$$\mathbf{F}_F = -F_{Fmag} \cdot \frac{\mathbf{v}_{relP}}{|\mathbf{v}_{relP}|} \quad (13)$$

By default the coefficients of friction are equal for all bodies. However, since different body pairs might interact the user can specify frictional coefficient for each contact pair individually.

## 4 Application of the Contact Library

### 4.1 General information about application

In order to enable collision handling in the *M.MB* library the components *CollisionWorld* and *CollisionBody* are needed. *CollisionBody* has a multibody connector frame that needs to be connected to a body respectively another multibody frame, see Figure 7.

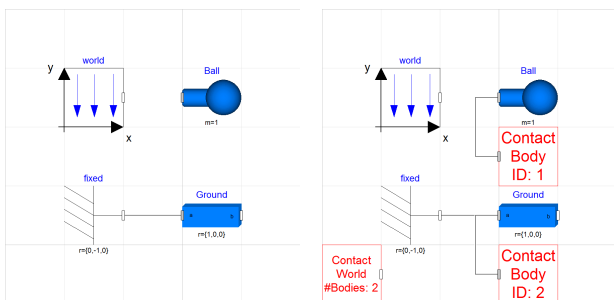


Figure 7: The model *BouncingBall* with and without collision consideration

Within the parameters of this component the user can specify the type of the collision shape and its geometrical parameters, see Figure 8.

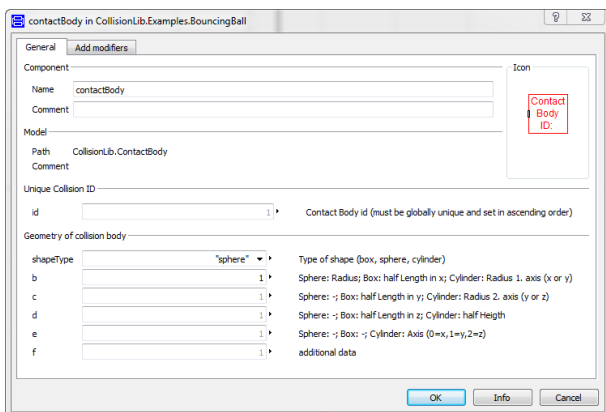


Figure 8: Parameters of the component *contactBody*

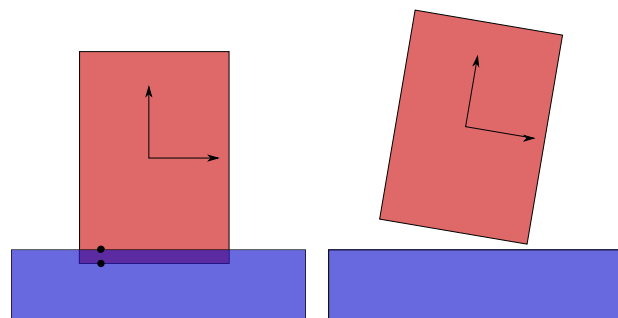
Each instance of *CollisionBody* needs a unique ID in ascending order starting at 1 (1,2,3,...). For the

*CollisionWorld* the highest ID has to be set as parameter in order to achieve "automatic" connections to the instances of *CollisionBody*. These connections are build using a *inner-outer-coupling*, meaning that an array of frame connectors is created for *CollisionWorld* with size of the passed parameter (maximum ID) and the *CollisionBody* is connected to the corresponding frame, based on the ID. Information about the shape is send from each instance of *CollisionBody* to the *CollisionWorld* via a user-defined connector. This is also done using *inner-outer-coupling*.

The problem concerning unique IDs has been discussed by Otter et al. in [1] before. However, planned changes in the Modelica language have not been implemented since then.

One of the bigger problems when handling contacts is the so called ghosting. This means, that one object moves through another object during one time step. This problem can only be solved by reducing the maximum solver step size. Alternatively within the *CollisionWorld* sampling can be activated, i.e. a collision check has to be performed during each sample. However, collision forces are also only calculated during each sample. This can result in wrong behavior, since the forces are constant between the samples.

The last big drawback when using the library *CollisionLib* are problems with collision detection for not strictly convex bodies. The GJK algorithm used by Bullet for many distance calculations, derives exactly one pair of deepest penetrating points, from the infinite number of pairs. This results in more or less unreal behavior, as pointed out in Figure 9. This issue



(a) configuration between cylinder and grounded box during contact with contact point pair (black)  
 (b) configuration after separation due to contact forces in the contact point pair

Figure 9: Wrong contact reactions due to an incongruous collision point pair

can be fixed by changing from deepest point penetration to intersection volume. However, a solution using the *Bullet Physics Library* is not known.

For some reason a user might want to ignore collision between a specific set of bodies. This feature is implemented by adding contact pairs (IDs) into a table within the component *CollisionWorld*.

### 4.2 Application 1: The pool table

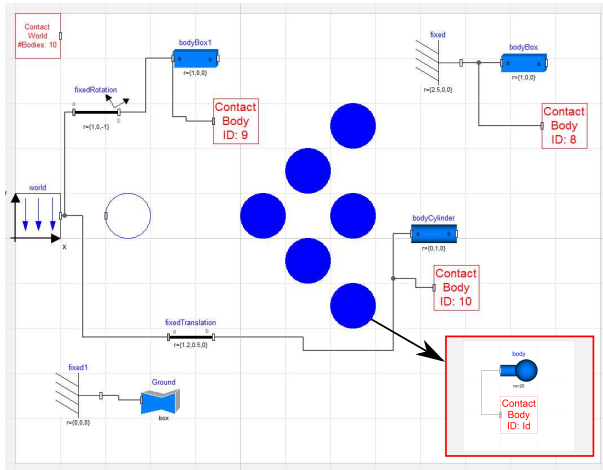


Figure 10: Setup of the model "billiard balls"

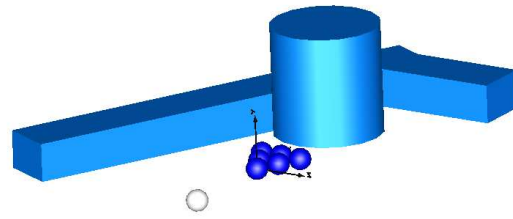
In Figure 10 a setup containing some "billiard balls" and other obstacles can be seen. While the obstacles are attached to the ground, the individual balls are free. Gravity of the system is set to zero. The white ball crashes into the group of balls, causing multiple collisions within the group of balls and the obstacles. In Figure 11 the system at time  $t = 0\text{ s}$ ,  $t = 0.25\text{ s}$  and  $t = 1\text{ s}$  is shown.

Note that the conservation of momentum is not fulfilled in the system. This Error occurs due to the chosen penalty approach.

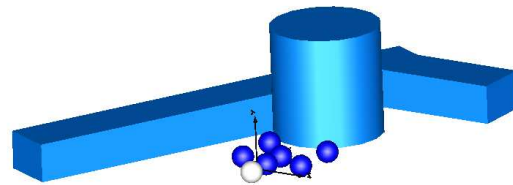
The computation time for this rather simple model is lower than the time simulated, which means for small models real-time simulation is possible.

### 4.3 Application 2: dynamically loaded compressing spring

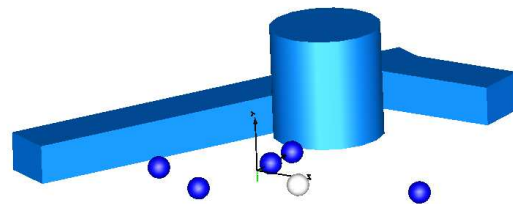
Apart from made-up models, the *CollisionLib* library has been used for the simulation of compressing springs. If simulating springs in high dynamic systems, like hydraulic valves, they can no longer be treated by the relation  $F = c \cdot \Delta x$ . Effects like coils hitting other coils or coils lifting of the spring cups need



(a) The system at time  $t = 0\text{ s}$



(b) The system after multiple collisions at time  $t = 0.25\text{ s}$



(c) The system after multiple collisions at time  $t = 1\text{ s}$

Figure 11: The model "billiard ball" with multiple collision partners

to be considered. Using collisions this consequences can be handled.

In picture 12 the multibody representation of the spring is shown. It consists of multiple rigid bodies that are connected by spring-damper-elements. There are no joints connecting the rigid spring elements or the single elements to the spring cap.

Due to a force of  $200\text{ N}$  acting for  $0.01\text{ s}$  on the upper spring cap, the spring is compressed and the coils interact among one another. At time step  $t = 0.03\text{ s}$  the spring has almost reached its block length and starts to expand shortly after. At  $t = 0.07\text{ s}$  the elements moved beyond their starting position during expansion of the

spring, see Figure 13.

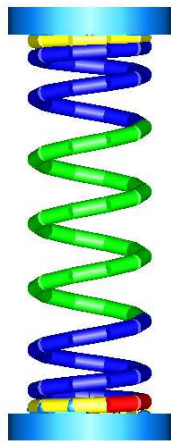
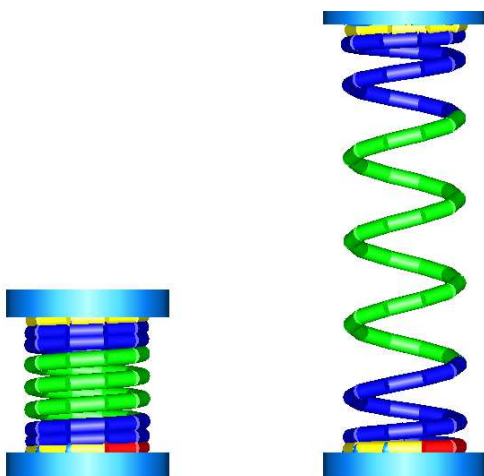


Figure 12: setup of a compressing spring model using the *CollisionLib* ( $t = 0$  s)



(a) The spring at  $t = 0.03$  s      (b) The spring at  $t = 0.07$  s

Figure 13: States of the spring during simulation

Highly detailed modeling of systems, like the spring with a lot of self-interactions, can benefit from collision handling. However, simulation times can become very large. The spring model shown here takes seven hours for the complete simulation. Simulation time will be a subject of further investigation.

## 5 Outlook

In this paper the development of an add-on library to the *M.MB* for consideration of collision handling was given. The present version can be used to improve

simulation, as shown in 4.3. However, there are some greater drawbacks that hinder the full potential of contacts in this early version of *CollisionLib*. Development in order to provide a fully functional collision library continues:

- Support for OpenModelica
- Enable calculation of more than one contact point pair within a contact.
- Finding a suitable solution for the ghosting effect (dynamically change of solver step size)
- Support of mesh representation as geometry information

## 6 Annotation

Help and cooperation to promote this project are very welcome. Feel free to send your ideas and offers to [Andreas.Hofmann7@boschrexroth.de](mailto:Andreas.Hofmann7@boschrexroth.de)

## References

- [1] Otter, M. et al. Collision Handling for the Modelica MultiBody Library. In: Proceedings of the 4th Modelica Conference 2002, Hamburg, Germany, Modelica Association, 7-8 March 2005.
- [2] Engelson, V. Integration of Collision Detection with Multibody System Library in Modelica. Linköping, Sweden: Thesis, Department of Computer and Information Science, Linköping University, 2000.
- [3] Baraff, D. Dynamic Simulation of Non-Penetrating Rigid Bodies. Ithaca, USA: PhD thesis, Department of Computer Science, Cornell University, 1992.
- [4] Mirtich, B. V. Impulse-based Dynamic Simulation of Rigid Body Systems. Berkeley, USA: Ph.D. thesis, Graduate Division, UC Berkeley.
- [5] Glocker, Ch. Dynamik von Starrkörpersystemen mit Reibung und Stößen. In: VDI-Fortschrittsbereiche Mechanik/Bruchmechanik, 1995
- [6] Beitelshmidt, M. Reibstöße in Mehrkörpersystemen. Munich, Germany: Ph.D. thesis, Lehrstuhl B für Mechanik, 1998.

- [7] Schmitt, A. A. Dynamische Simulation von gelenkgekoppelten Starrkörpersystemen mit der Impulstechnik. Karlsruhe, Germany: internal report, Department of Computer Science, Karlsruhe Institute of Technology, 2003.
- [8] Bullet Physics Library. available at: [www.bulletphysics.com](http://www.bulletphysics.com).
- [9] Coumans, E. Bullet 2.80 Physics SDK Manual. URL: <http://www.cs.uu.nl/docs/vakken/mgp/assignment/Bullet%20-%20User%20Manual.pdf>, 2012.
- [10] Machado, M. et al. Compliant contact force models in multibody dynamics: Evolution of the hertz contact theory. In: Mechanism and Machine Theory 53 (2012), pages 99-121.
- [11] Gilbert, E.G., Johnson, D.W. and Keerthi, S.S. A fast procedure for computing the distance between complex objects in three-dimensional space. In: IEEE Journal of Robotics and Automation 4(2), pages 193-203, 1988.
- [12] Cohen, J.D. et al. I-Collide: An Interactive and Exact Collision Detection System for Large-Scale Environments. In: Proceedings of the 1995 Symposium on Interactive 3D Graphics 189ff, Monterey, CA, USA, 1995.



# Short-term production planning for district heating networks with JModelica.org

Stéphane Velut<sup>1</sup>, Per-Ola Larsson<sup>1</sup>, Johan Windahl<sup>1</sup>, Linn Saarinen<sup>2</sup>, Katarina Boman<sup>2</sup>

<sup>1</sup> Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

<sup>2</sup> Vattenfall R&D, 814 26 Älvkarleby, Sweden

stephane.velut@modelon.com, per-ola.larsson@modelon.com

## Abstract

The short term thermal production planning problem is solved in two steps by integrating physical plant models into the standard approach. The first step aims at solving the discrete variables from the unit commitment sub-problem (UCP) using standard mixed integer linear models and optimization techniques. The second step focuses on the economic dispatch sub-problem (EDP) described by high-fidelity, continuous time, physics-based Modelica models together with nonlinear optimization techniques from the JModelica.org platform. The output of the second step includes optimized power flows but also highly relevant variables such as supply temperature, supply flow rate, turbine by-pass valve in the cogeneration plant. The optimization is formulated as a maximization of the benefit from heat and electricity sell over a finite time-horizon.

The proposed method is validated in several test cases using experimental data from a plant in Nyköping. The optimizations demonstrate the feasibility and the high economic potential of the proposed approach when comparing with measurement data and the standard optimization techniques. The optimized planning schedules result in a balance between produced and consumed heat, priority to low-cost boilers and maximization plant revenue. Compared to measurement data, the optimizations result in a significantly lower supply temperature, a more extensive usage of the external cooler for higher efficiency and higher electricity production, fewer starts of units as well as an appropriate use of the accumulator tank.

The high-level description of optimization problems using JModelica.org provides useful means to speci-

fy flexible optimization problems including constraints on arbitrary process variables such as heat load of the production units, supply temperature and flow rate, pressures.

*Keywords: production planning; nonlinear optimization; district heating; physical modeling; unit commitment*

## 1 Introduction

### 1.1 Background

#### 1.1.1 Production planning

Production planning in district heating systems aims at finding a cost optimal scheduling of the heat and power production plants, which satisfies both the network load demand and operational constraints. Scheduling refers to the status of the production unit (on-off, a discrete variable), and the produced power (a continuous variable). The resulting optimization problem that involves both discrete and continuous variables is referred to as mixed integer non-linear problem (MINLP), for which no robust algorithm is available. It is therefore necessary to make reasonable assumptions on both the modeling and the computation approaches to get a tractable optimization problem. The problem can be described as being composed of two sub-problems:

- The Unit Commitment Problem (UCP), in which decisions are taken on whether a plant should be running or not. The main difficulty lies in the combinatorial nature of the problem.
- The Economic Dispatch Problem (EDP) in which the load decisions for all active plants are taken. The main difficulty lies in the nonlinearity of the plant.

A good survey of the available approaches for short-term production planning is given in [15].

As an input to the production planning problem, a predicted heat load over the entire optimization horizon should be provided. This is often generated by a load prediction model that typically includes a description of the district heating network and the effect of outdoor temperature. Here it is assumed that a perfect load prediction is available over the entire optimization horizon.

### 1.1.2 Common approach

The standard formulation of the thermal production planning problems relies on a simplified representation of the model equations. The plant models are typically linearized and the resulting problem becomes a Mixed Integer Linear Program (MILP). The continuous decision variables of the optimization problem are the energy flows whereas the influence of the supply temperature and mass flow are usually not modeled. This represents a limitation since e.g., supply temperature affects many critical parameters such as the amount of energy that can be stored in the network or the accumulator, the heat loss in the network and the electric efficiency of the co-generation plants. To maintain a low model complexity, it is also common to describe all processes by static relationships except for the storage dynamics (heat and fuels) and eventually the transport delays in the distribution network. The linearization process, which is a necessary and critical part of the MILP approach, is consequently a trade-off between model accuracy and tractable model complexity.

## 1.2 Proposed approach

The proposed approach is based on the natural separation of the discrete problem (UCP) from the continuous one (EDP).

- UCP. The entire optimization problem is formulated using simple piecewise linear models and solved using a MILP solver. The main result of this stage is the status of every plant (on/off) over the optimization horizon.
- EDP. The desired load is dispatched between the running production plants to meet all plant operational and safety constraints. The status (on/off) of the plants and the start values of continuous control signals are given by the solution of the UCP.

The aim of the second step is to optimize the non-linear plant model based on physical laws without

any major simplification. The plants are described by mass and energy balances, in terms of enthalpy, mass flow rate and pressure. Dynamics can be included without restrictions to match the real dynamic behavior of the plants. The output of this second step includes optimized power flows, but also highly relevant variables that affect the production economy such as supply temperature, supply flow rate or turbine by-pass valve in the co-generation plant. This model complexity yields however a non-linear dynamic optimization problem and requires another type of solver than MILP solvers, see [1] for an overview of the available strategies. One reliable and efficient method to solve dynamic optimization problems that is based on non-linear programming solvers is the so-called collocation method. Control signals to be optimized and model equations are parameterized by a smaller number of variables, reducing considerably the complexity of the non-linear optimization problem. The original continuous-time optimization problem is transformed into a (discrete-time) Non-linear Programming (NLP) problem that can be efficiently solved using commercial or open-source solvers. The authors have applied the collocation method for dynamic optimization of a Carbon Capture plant, see [2]. Other successful applications of this optimization technique have been reported in the literature, see [3] for a list of applications where IPOPT (Interior Point Optimizer), an open-source NLP solver, was used. To the authors' knowledge, limited work on applying large-scale NLP methods for solving the economic dispatch problem has been performed.

## 2 Optimization tools and languages

Two environments were used to define and solve the production planning problem:

- Dymola [9] was the chosen platform to derive, calibrate and simulate the physical Modelica [10] models defining the economic dispatch sub-problem
- Python was the chosen platform to solve both optimization problems and to do the post-processing

The unit commitment problem (UCP) was formulated in Python using the package PuLP [4] that is a light weight package that allows modelers to easily express mathematical programming problems, including mixed integer linear programs. It uses a high level modeling language and has been built to interface with commercial and open-source solvers. In the present work the solver CBC has been used, see [5].



The economic dispatch problem was solved from Python using the JModelica.org platform [16] as follows:

- The optimization problem was formulated in the Modelica and Optimica languages
- An FMU-simulation was carried out to generate a guess trajectory for the initialization of the nonlinear optimization
- The nonlinear optimization problem was solved using a collocation method available in JModelica.org

Applying nonlinear dynamic optimization techniques to solve the economic dispatch problem requires some considerations on initial trajectories, smooth media model functions, smoothing of mathematical functions such as absolute value, square root, min and max as well as scaling of optimization variables yielding a numerically sound behavior.

### 3 Plant models

#### 3.1 Idbäcken plant

The Vattenfall AB owned district heating plant Idbäcken, providing heat to customers in Nyköping, Sweden, served as reference plant. It contains in total seven different heat production units, one accumulator and one co-generation plant, see Figure 1. The Idbäcken plant has been the topic of interest in other research projects, see for instance [6] [7] and [8].

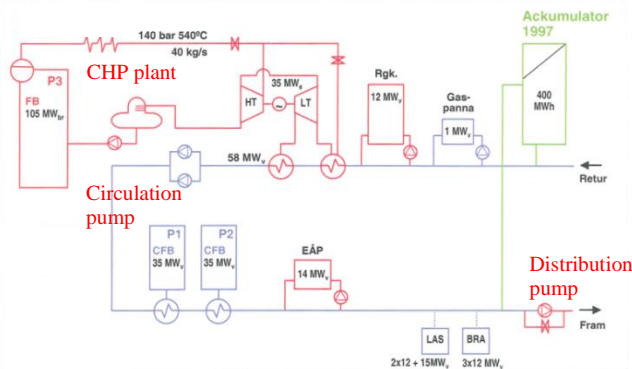


Figure 1. Schematic diagram of the Idbäcken plant with 7 heat production units, the accumulator and the co-generation plant

The main unit of Nyköping is the Combined Heat and Power plant (CHP) with a total heat capacity of 105 MW. The plant turbines can at most produce 35 MW of electricity, and the condensers can approximately transfer 75 MW of heat to the district heat water. The flue gas of P3 is used in a flue gas con-

denser of maximum 12 MW to pre-heat district heat return water before entering the CHP plant.

Two circulating fluidized bed units, P1 and P2, of each 35 MW used only for heat production may be used when heat demand is high. Additionally, there is an electric boiler of 14 MW and two oil based heat production units at the hospital and the residential area Brandkärr with 2×12 and 3×12 MW production capacity, respectively.

There is also an accumulator tank of hot water and an external cooler Beriden (not shown in figure) that can be used to decrease the return temperature and therefore increase both the load and the efficiency of the co-generation plant for a higher electricity production.

#### 3.2 Models for UCP

The unit commitment problem is formulated in Python using the PuLP modeling language.

##### 3.2.1 Heat production units

The discrete time plant model, i.e., the UCP model, is very coarse compared to the EDP model. All heat producing units  $i$ , except for the CHP plant, are modeled only by the produced heat  $Q_i$  and its minimum and maximum capacity. The influence of plant actuators such as pumps, valves or the influence of physical variables such as temperature, pressure and mass flows are not captured. The fuel consumption  $U_i$  of a pure heat unit is calculated using the efficiency parameters  $\eta_i$  as

$$U_i = \frac{Q_i}{\eta_i}$$

and for the CHP plant it is calculated as

$$U_3 = \frac{Q_{P3} + P_{el}}{\eta_3}.$$

For the CHP plant also the ratio between produced electricity and heat is interesting, defining the  $\alpha$  value as,

$$\alpha = \frac{P_{el}}{Q_{P3}}$$

The accumulator energy  $E_{acc}$  is given by a simple integrator equation and is not temperature dependent:

$$E_{acc}[t] = E_{acc}[t - 1] - hQ_{acc}[t - 1]$$

where  $Q_{acc}[t]$  is the energy flow from the accumulator,  $h$  is the sample time and  $t$  is the time index.

### 3.2.2 Co-generation plant P3

The produced heat  $Q_{P3}$  and electric power  $P_{el}$  are influenced by the boiler load, the position of the turbine by-pass valve as well as the flow and temperature of the district water. It is by far too complex to describe this relationship by simple (piecewise) linear functions. By using a physics based Modelica plant model and varying the four main variables, a feasible region in the  $Q_{P3}$ - $P_{el}$ -plane has been created to describe the behavior of the cogeneration plant. The resulting feasible region was described by four inequalities of the form

$$a_i P_{el} + b_i Q_{P3} + c_i \geq 0$$

where the points  $a_i$ ,  $b_i$  and  $c_i$  are scalar coefficients.

The UCP optimization should optimize  $Q_{P3}$  and  $P_{el}$  with the constraint that they should be in the feasible region described by the inequalities.

### 3.3 Models for EDP

The models used to formulate the economic dispatch problem are implemented in Modelica.

#### 3.3.1 Medium model

Two different types of media models are used in the model package:

1. Simple water media – a water media with constant specific heat capacity and density, used to describe the water in the district heating network.
2. Advanced water media – a water media with polynomial functions approximating IF97 reference functions, used to describe liquid and vapor water in the cogeneration plant, see [17] for more details.

#### 3.3.2 Cogeneration plant

The co-generation (heat and electricity production) unit P3, shown in Figure 2, is described by the following models

- One high pressure turbine and two low pressure turbines, all with bleed streams
- Two condensers
- By-pass valve for by-passing steam directly to the condensers
- A lumped bleed streams to describe the flow to the de-aerator and the high-pressure pre-heater
- Control volumes

The characteristics that are important to capture with respect to the optimization is the influence of the turbine by-pass valve, the district heat network flow and temperature and the boiler load on the produced heat and electricity. It is not necessary to describe e.g., the furnace, instead, focus has been directed towards the vapor cycle. The main modeling simplifications on the vapor cycle is that it is not closed, which results in the following assumptions:

- The vapor characteristics (pressure and enthalpy) at the boiler outlet are constant and the boiler load linearly affects the boiler mass flow rate.
- The feed water heaters that are downstream of the condensers are not modeled. All the bleed streams that normally go to the non-modeled pre-heaters are represented by a single stream connected to a constant pressure source.
- The condensate leaving the condenser is assumed to be at saturation pressure.

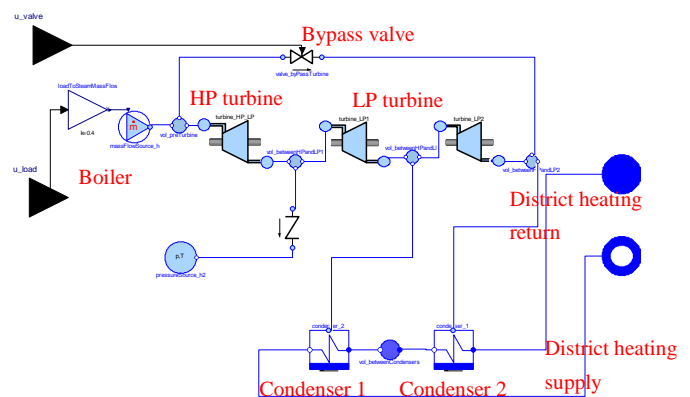
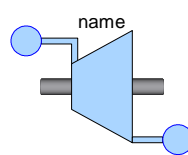


Figure 2. Dymola model of the cogeneration plant P3 with connections to the district water.

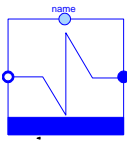
#### 3.3.3 Fluid component models

##### 3.3.3.1 Turbine

Physics-based model defined by an isentropic efficiency to calculate the outlet enthalpy and turbine work. The mechanical power generated from the steam is calculated using a mechanical efficiency and the pressure drop is related to the flow rate using Stodola's law. The electric power is calculated from the mechanical power using an efficiency parameter (generator losses).

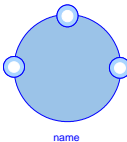


### 3.3.3.2 Condenser



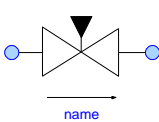
The heat flow rate transferred to the district water is driven by the temperature difference between the incoming water and the saturation temperature in the condenser. This heat flow rate is further used to compute the condensation rate that drives the bleeding flow from the turbine.

### 3.3.3.3 Control Volume



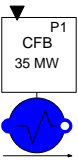
The control volume is a straightforward implementation of dynamic mass and energy balances expressed using pressure and enthalpy as states. Temperature is computed using pressure and enthalpy. The model requires partial derivatives of density with respect to enthalpy and pressure.

### 3.3.3.4 Valve and pressure loss



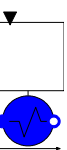
The mass flow through the valve is computed using the pressure difference, the valve opening and data from a nominal point. A standard quadratic equation relates mass flow and pressure drop.

### 3.3.3.5 Circulating Fluidized Beds, P1 and P2



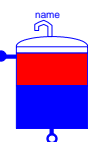
The model is not physics-based and heat transferred to the water is calculated by first-order filtering of the load and using parameters for efficiency and maximum heat transfer.

### 3.3.3.6 Electric Boiler and Oil Driven Boilers



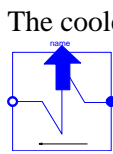
The model is not physics-based and heat transferred to the water is calculated by the load and using parameters for efficiency and maximum heat transfer.

### 3.3.3.7 Accumulator



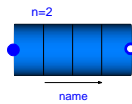
The accumulator is modeled using a finite-volume approximation that neglects buoyancy effects, i.e. no mixing is assumed when the accumulator is not charging or discharging. The accumulator is charged and discharged from the top and bottom. Return water enters from the bottom. Heat loss has been neglected.

### 3.3.3.8 External cooler



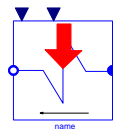
The cooler is modeled as a lumped epsilon-NTU heat exchanger where heat transfer is driven by the difference between the inlet temperatures. It is assumed that the minimal heat capacity flow is always on the district side flow.

### 3.3.3.9 Transport Pipe



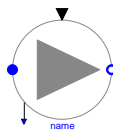
It is a finite volume implementation of a pipe with control volumes in series. The nominal number of discretization segments is 4 and the pipe diameter and length are parameters.

### 3.3.3.10 Flue Gas Condenser



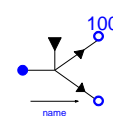
The model is not physics-based and is an ideal heat source which produces constant heat as long as P3 is running.

### 3.3.3.11 Pump



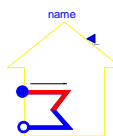
The pumps are ideally modeled, delivering a specific mass flow rate depending linearly on the control input, i.e., pump speed.

### 3.3.3.12 Splitter/Valve



The splitter/valve is ideally modeled and splits the incoming flow into two flows which sizes depend on the control signal to the splitter.

### 3.3.3.13 Customer



The customer is modelled with a fixed outlet temperature and the heat absorbed is calculated by the temperature difference between in- and outlet and the flow that is merely propagated through from in- to outlet.

## 4 Problem formulation

### 4.1 Degrees of Freedom

The discrete time optimization (UCP) includes the power heat flows, the electricity production rate as well as the status (on/off) of every unit as optimiza-

tion variables. The status of each unit is fixed in the continuous optimization.

The continuous time optimization problem contains 11 degrees of freedom, namely:

- $U_i(t)$  fuel load of production unit  $i$ ,  $1 \leq i \leq 6$
- $U_{BP}(t)$ , turbine by-pass valve position
- $U_{CP}(t)$ , Speed of circulation pump
- $U_{DP}(t)$ , Speed of distribution pump
- $U_{split}(t)$ , ratio between flows going to customer and by-passing customer respectively
- $U_{cooler}(t)$ , ratio between flows going to external cooler (Beriden) and by-passing it

In the optimizations however, the decision variables used by the optimization routine will be the derivatives of the above inputs. Thus, an equation on the form

$$U_j(t) = \int_t \dot{U}_j(t) dt$$

is introduced for each input. This extension makes it easy to set minimum and maximum constraints on the input signal derivatives in the optimizations. In Modelica code, for a general input  $u$  with minimum and maximum values  $u_{min}$  and  $u_{max}$  and minimum and maximum derivative values  $\dot{u}_{min}$  and  $\dot{u}_{max}$ , an example is the following:

```
model inputExample
  parameter Real u_min = -1;
  parameter Real u_max = 1;
  parameter Real u_der_min = -1;
  parameter Real u_der_max = 1;
  input Real u_der(min = u_der_min,
max = u_der_max);
  Real u(min = u_min, max = u_max);
equation
  der(u) = u_der;
end inputExample.
```

Another possibility is to utilize the constraint-section in the optimization model.

## 4.2 Cost function

The goal of the optimization, both in the UCP and EDP, is to produce enough heat to follow the customer heat load over time and at the same time do it as economically beneficial as possible. The EDP problem considers fuel costs  $U_i(t)p_i$  and incomes from selling heat  $P_{el}(t)p_{el}$  and electricity  $Q(t)p_Q$  to describe the plant economy. Note that operational costs for e.g., pumps are not considered. The cost for

starting a production unit  $start_i[t]s_i$  is only included in the objective function of the UCP sub-problem. Hence, the revenue at a time instant can thus be formulated as

$$R(t) = P_{el}(t)p_{el} + Q(t)p_Q - \sum_i U_i(t)p_i - start_i(t)s_i,$$

A minor cost on the input derivatives must be used for regularity reasons. This cost at a certain time instant is formulated as

$$\dot{W}(t) = \sum_{\substack{j=All\ model \\ inputs}} q_{\dot{U}_j} \dot{U}_j(t).$$

where  $q_{\dot{U}_j}$  is the weight for derivative  $\dot{U}_j$ .

The cost function to be minimized, considering the cumulative revenue over the optimization horizon, can thus be formulated as

$$J = \int_{t_0}^{t_f} (\dot{W}(t) - R(t)) dt$$

where the optimization interval is  $[t_0, t_f]$  and 24 h long. This is as long as the optimization horizon in the UCP and also as the prediction time series provided by the heat load prediction model. Without the extra term of the input derivatives, this cost function is the continuous time counter part of the cost function in the UCP except for the start-up costs.

## 4.3 Constraints

### 4.3.1 Common constraints for EDP/UCP

For fulfillment of the heat demand from the customers, the following constraint is used in both EDP and UCP:

$$-Q_{dev.max} \leq Q(t) - Q_d(t) \leq Q_{dev.max}$$

where  $Q$  is the heat delivered to the customers,  $Q_d$  the desired heat load from the prediction model. The maximum deviation  $Q_{dev.max}$  is set to 1 MW. The supply temperature is not a direct function of the outdoor temperature, which is common practice. Instead, the supply temperature depends on the supply flow and heat demand, where the latter is a prediction depending on predicted outdoor temperature. A lower bound on the supply temperature has also been introduced and this could be replaced by a lower bound on the temperature at the customer substation if the district heating network is modelled.

An accumulator energy end-point constraint is introduced in both EDP and UCP to avoid that the accumulator gets empty at the end of the optimization interval:

$$E_{acc}(t_f) \geq E_{acc}(t_0)$$

The input derivatives, i.e., the decision variables for the optimizer, are given minimum and maximum values. A maximum change of  $\pm 2\%/min.$  has been set for all input derivatives relative to the maximum value of the input to the model. These limits can be set both from a physical perspective but also from a numerical perspective such that highly changing control signals are avoided. In the optimization results presented in this report, the derivative constraints are far from being active.

#### 4.3.2 UCP specific constraints

The start and stop of large solid fuel boilers is time-consuming and it is therefore critical to model the delay between the start/stop decision and the time the boiler is running at minimum/zero load. All units that are starting or stopping are required to follow a pre-defined trajectory  $Q_{i,start}[t]$  or  $Q_{i,stop}[t]$ , respectively. The constraints are formulated as

$$\begin{aligned} Q_i[t] &= Q_{i,start}[t], \\ t &\in [t_{start}, t_{start} + t_{startdelay}] \\ Q_i[t] &= Q_{i,stop}[t], \\ t &\in [t_{stop}, t_{stop} + t_{stopdelay}] \end{aligned}$$

where  $t_{startdelay}$  and  $t_{stopdelay}$  are the lengths of start and stop sequences in time. The details on how the timing variables  $start_i[t]$ ,  $stop_i[t]$ ,  $starting_i[t]$  and  $stopping_i[t]$  relate to the starting and stopping sequence equations above can be found in [11].

The accumulator storage capacity and heat flow are constrained with minimum and maximum values as

$$\begin{aligned} E_{acc,min} &\leq E_{acc}[t] \leq E_{acc,max} \\ Q_{acc,min} &\leq Q_{acc}[t] \leq Q_{acc,max} \end{aligned}$$

#### 4.3.3 EDP specific constraint

Normally, the supply temperature is chosen as a function of the outdoor temperature. In the continuous optimization, the supply temperature is optimized and allowed to vary between  $74.5^\circ\text{C}$  and  $110^\circ\text{C}$ . Also the flow to the customers, essentially the distribution pump flow, is given minimum and maximum values. These are  $0 \text{ kg/s}$  and  $550 \text{ kg/s}$ .

## 5 Optimization example

Different test cases based on measurement data from Idbäcken plant were considered to evaluate the production planning strategy. All optimizations, both UCP and EDP, are performed using an optimization horizon of 24 h. The UCP sampling time is 0.5 h while the number of elements in the collocation scheme is 72, i.e., the length of each element is 20 minutes. One test case with two load peaks is presented in the paper.

### 5.1 Initialization

All dynamic models were initialized using experimental data of the real plant. The variables of the UCP model that require initialization are the status of each unit, the heat and electricity production of the co-generation plant as well as the accumulator energy. The physical model of the plant used to formulate the EDP sub-problem contains much more states to be initialized. The start values were either directly taken from the measurement data or computed using an FMU simulation of the plant.

### 5.2 Computational statistics

In all optimization cases, the UCP optimization formulation results in a MILP of approximately 4000 decision variables and 7000 constraints. The sampling time considered is 30 minutes and the optimization horizon is 24 h. With the CBC MILP solver, see [5], the solution time is less than 25 s for all optimization cases.

The resulting NLP, after discretization of the EDP optimization problem using 72 elements each of a length 20 minutes, is solved using IPOPT v3.10.0 running with the linear solver MA27, see [12] and [13]. The NLP contains approximately 130 000 variables for each of the considered optimization cases, which can be considered a small to medium sized problem for IPOPT. Much larger problems have been solved using the JModelica.org framework and IPOPT, see [14] and references therein. The solution time is approximately 2-5 minutes and depends on e.g., initial guesses number of decision variables and number of active constraints.

### 5.3 Specific test case

#### 5.3.1 Experimental data

In this test case, the co-generation plant P3 is the only running unit at the beginning of the considered

time interval. The heat load profile showed in the figure below displays two peaks that require the start of additional units. To meet that increasing customer demand, the heat production in the cogeneration plant was first prioritized by fully opening the bypass valve before both P1 and P2 were started, see dotted curves in Figures 4 and 5.

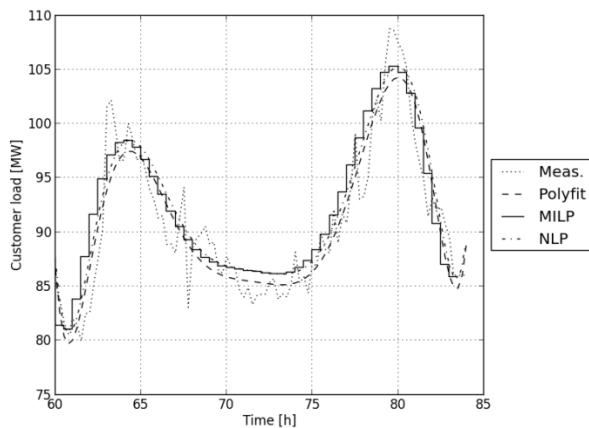


Figure 3. Profile of the customer heat load: measurement, polynomial approximation, MILP and NLP solutions.

### 5.3.2 Optimization results

The solution of the unit commitment problem is characterized by the start of P2 at the first sample to meet the increasing customer demand. The start of P1 does not seem to be necessary. As the first peak of the customer heat demand comes before P2 is fully available, the accumulator is used and its energy level is significantly decreased during the first half of the optimization interval. P2 stays at its minimum capacity during the first 12h, a stop and re-start would be a more expensive alternative. During the second half of the optimization, P2 is used to load the accumulator and provide heat to the network. The optimization leads to a constant and maximum heat electricity production as it is economically beneficial. The turbine by-pass valve is therefore kept closed during the entire optimization interval. The variation in the customer load is met by changing the load in P2 and by using the accumulator. Figure 6 shows one typical advantage of applying nonlinear optimization for the economic dispatch: the supply temperature and flow are optimized to maximize the benefits. Note that none of these variables are in the UCP formulation as the UCP only contains heat and energy variables. For minimal fuel consumption, the supply temperature is kept low, about 5 degrees lower than in measurement data, the flow rate to the

network is instead increased to meet the heat power demand. Note that the distribution pump operates close to its maximal capacity except when the accumulator needs to be charged. Accumulator loading requires indeed a difference in the flows delivered by the circulation and the distribution pumps. At  $t=70h$ , the circulation pump operates at its maximum and the distribution pump is therefore forced to be decreased for accumulator loading.

### 5.4 Conclusion from other test cases

The production planning strategy has been successfully tested and compared with experimental data in 6 different cases, see [18] for more details. The following conclusions can be drawn:

- It is fully possible to integrate nonlinear optimization techniques from JModelica.org into the standard production planning approach for more accurate and more informative production plans.
- One of the main advantage of combining physical modeling and nonlinear optimization techniques is the optimized supply temperature and mass flow rate as well as a more accurate description of the accumulator.
- In the considered network, the optimization showed a high potential income related to the use of the external cooler that reduces the return temperature for a higher electric efficiency and a higher electricity production. This only occurs at low customer loads, i.e., when P3 is not required to run on full load. If customer load is high, then P3 should already be running at full load and the electricity production maximized.
- Handling of constraints on physical variables such as temperature, pump capacity and also constraints related to the way the accumulator is connected, improves substantially the quality of the optimization results.
- Supply temperature can be decreased by approximately  $4^{\circ}C$  compared to experimental data. With a perfect load prediction, the savings compared to measurement data is about 8%, which is related to the extensive use of the external cooler.

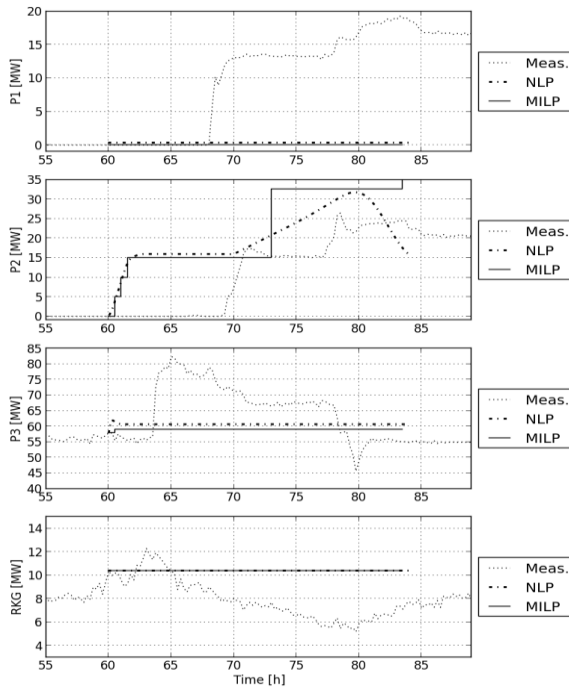


Figure 4. Heat production of the main units: measurement (dotted), MILP (solid) and NLP (dashed).

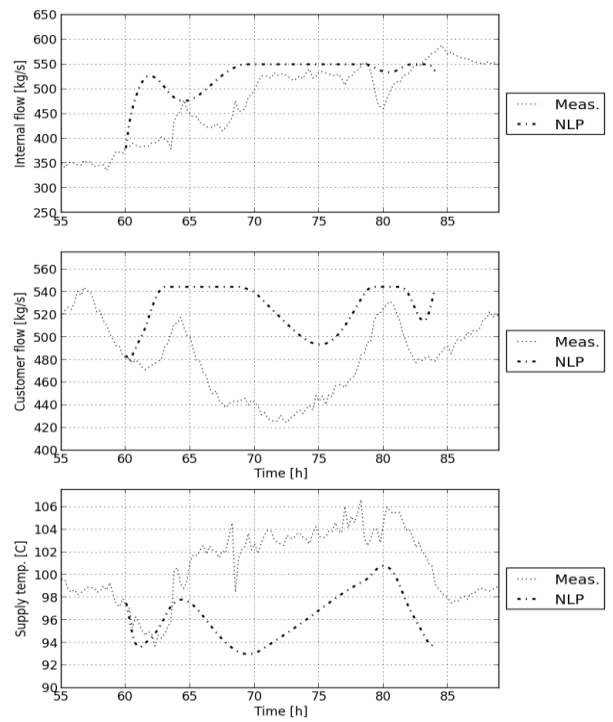


Figure 6. Circulation flow, flow to network and supply temperature. Measurement (dotted) and NLP (dashed).

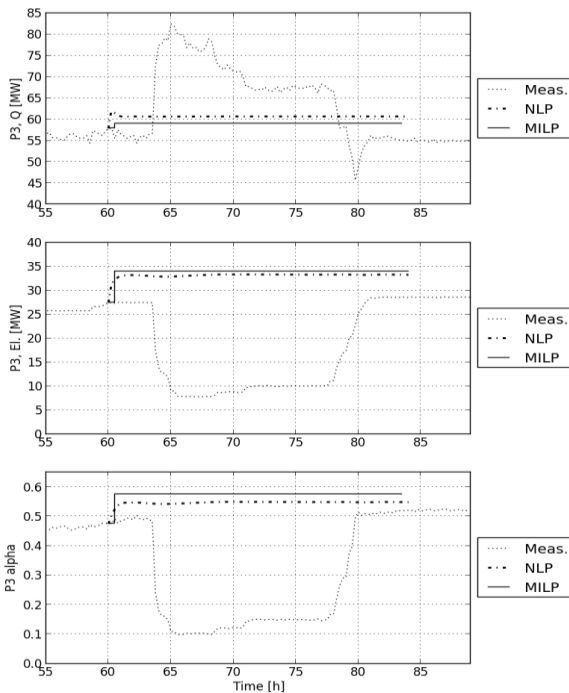


Figure 5. Heat, electricity production rates and alpha-value of the cogeneration plant P3.

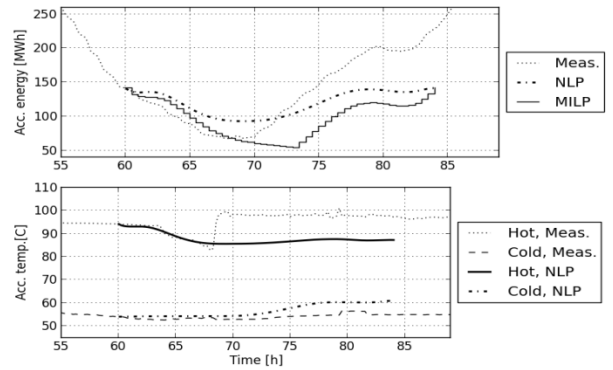


Figure 7. Accumulator energy and temperatures.

## 6 Conclusions

A method that substantially improves the standard approach for short-term production planning has been proposed in the paper. It is based on the use of physical plant models and nonlinear optimization techniques from JModelica.org to solve the economic dispatch sub-problem. The strategy has been validated using experimental data from a plant in Nyköping.

ping Sweden. Compared to measurement data, the method results in a significantly lower supply temperature, a more extensive usage of the external cooler for higher efficiency and higher electricity production. The flexible optimization platform JModelica.org makes it possible to optimize the plant economy and introduce constraints on critical variables such as temperature, pressure or flow.

## 7 Acknowledgement

Värmeforsk, "The Swedish Thermal Engineering Research Institute" and Energimyndigheten "The Swedish Energy Agency" are gratefully acknowledged for their financial support (project P12-203).

## References

- [1] C. Cervantes and L. T. Biegler, "Optimization strategies for dynamic systems," in C. Floudas, P. Pardalos (Eds), *Encyclopedia of Optimization*, 2000.
- [2] J. Åkesson, C. Laird, G. Lavedan, K. Prölss, H. Tummesheit, S. Velut and Y. Zhu, "Nonlinear Model Predictive Control of a CO<sub>2</sub> post-combustion unit," *Chemical Engineering Technology*, 2011.
- [3] Ipopt homepage, coin-or, <http://projects.coin-or.org/Ipopt/wiki/IpoptPapers>.
- [4] S. Mitchell, A. Mason, M. O'Sullivan and A. Phillips, "PuLP: a linear programming toolkit for python," <http://www.coin-or.org/PuLP/>.
- [5] CBC Team, "CBC home page," 2013. [Online]. Available: <https://projects.coin-or.org/Cbc>. [Accessed 12 August 2013].
- [6] L. Saarinen, "Model-based control of district heating supply temperature," Värmeforsk P08-819, 2010.
- [7] L. Saarinen and K. Boman, "Optimized district heating supply temperature for large networks," Värmeforsk P08-830, 2012.
- [8] L. Saarinen, "Modeling and control of a district heating system," Master thesis, Uppsala University, 2008.
- [9] Dassault Systemes, "Dassault Systemes Home Page," 2013. [Online]. Available: <http://www.3ds.com/products/catia/portfolio/dymola>. [Accessed 6 August 2013].
- [10] The Modelica Association, "The Modelica Association Home Page," 2013. [Online]. Available: <http://www.modelica.org>. [Accessed 6 August 2013].
- [11] J. Arroyo and A. Conejo, "Modeling of start-up and shut-down power trajectories of thermal units," *IEEE Transactions on power systems*, vol. 19, no. 3, 2004.
- [12] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 196, no. 1, pp. 25-68, 2006.
- [13] HSL, "A collection of Fortran codes for large scale scientific computation," 2013. [Online]. Available: <http://www.hsl.rl.ac.uk>. [Accessed 13 August 2013].
- [14] P.-O. Larsson, PhD thesis: Optimization of Low-Level Controllers and High-Level Polymer Grade Changes, Lund, 2011.
- [15] E. Dotzauer, "Algorithms for Short-Term Production planning of Cogeneration Plant," Lic. Thesis, Linköping University, 1997.
- [16] "www.jmodelica.org," Modelon AB, 2013. [Online]. Available: [www.jmodelica.org](http://www.jmodelica.org). [Accessed 2013].
- [17] Bauer, O. Modelling of Two-Phase Flows with Modelica, Master's Thesis, Lund University, Department of Automatic Control, 1999.
- [18] S. Velut, P.O. Larsson, J. Windahl, L. Saarinen, K. Boman, Non-linear and Dynamic Optimization for Short-term Production Planning. Värmeforsk report, 2013.



# Modelling the system dynamics of islanding asynchronous generators

Håkon Molland Edvardson  
Oslo, Norway  
haakky1@gmail.com

Dietmar Winkler  
Telemark University College, Norway  
dietmar.winkler@hit.no

## Abstract

Asynchronous generators are often used for small hydro power stations with an installed power capacity of under  $1\text{ MW}$ . The reason for this is their robustness and low cost. In order to be able to produce active electrical power with an asynchronous generator one needs to provide enough excitation by means of reactive power provided by either the electrical grid or additional capacitors.

But in asynchronous generators we can also find the phenomenon of self-excitation which allows the asynchronous generator to operate as a standalone unit. Investigation of the self-excitation process shows that significant over-voltages can occur if a generator with sufficient capacitors is suddenly disconnected from the utility grid. The precondition for a successive voltage build-up is that the generator is left with enough capacitive power and a low load after the disconnection.

The *Lønnestad* radial in *Seljord, Norway*, is a distribution radial with both asynchronous and synchronous generators connected. In order to investigate the system dynamics in the radial after it is disconnected from the rest of the  $22\text{ kV}$  distribution grid, the radial was modelled and simulated using *Modelica* as modelling language.

**Keywords:** *modelica, asynchronous generators, self-excitation, islanding, electric power library*

## 1 Introduction

The main share of the electricity produced in Norway is based on utilisation of the nation's large potential of hydro power. Today are nearly all the large waterfalls profitable for hydro power production already utilised, or protected against encroachment on nature. Due to this, there has for the last decades been an expansion in the number of small hydro power stations below  $10\text{ MW}$ . This is often minor

projects where the power station is located near a small waterfall owned by a local landowner.

These small hydro power stations are often connected to already existing distribution grids, due to the geographical location and installed capacity of these stations. This is often grids constructed for low capacities with purpose to distribute the electricity out to the local consumers. Connection of power stations in these types of grids will therefore often change the situation of power flow in the grid, and lead to challenges regarding voltage stability and requirements for faults detection.

## 2 Theory

The system that will be described in this paper consists of a electrical distribution radial to which one synchronous generator and several asynchronous generators are connected. This theory section will only cover the most important aspects of the complex power systems.

### 2.1 Asynchronous generators

In the industry the asynchronous machine, or induction machine, is used in a wide variety of applications with purpose of converting electrical power to mechanical work. The asynchronous machine is very economical, reliable, and easy to control, which are some of the reasons for its popularity. There are two main types of asynchronous machines based on the rotor construction; squirrel cage type, and wound rotor type. The simplicity and low cost, and the fact that they can be driven as a generator as well as a motor, makes these machines very beneficial for wind power generation and small hydro power stations up to  $1\text{ MW}$ .

Unlike in wind power applications, the wound generator is seldom used in small hydro power stations. The reason for this is that generators used in small hydro power stations generally is operated on the

principle of self-excitation without any rotor excitation. For such applications, generators with the squirrel cage rotors can with advantage be used instead of generators with wound rotor, since the squirrel cage machine has lower cost.

### 2.1.1 The phenomena of self-excitation

Unlike the synchronous generator which gets its magnetisation from an internal magnetising source, and can be controlled to operate at a given frequency, the induction generator has no independent control over the air-gap field. The induction generator needs lagging reactive power to produce the main air-gap and winding leakage flux [1]. This phenomenon is referred to as self-excitation since the generator achieves its magnetising from a grid, or capacitors which are connected to the stator terminals. The phenomena permit utilisation of an induction generator as a standalone unit without a voltage source connected. Due to this the induction generator is often referred to as a *SEIG* (Self-Excited Induction Generator).

The phenomenon of self-excitation has been known for a long time, and a great deal of research has been done in the field of describing the phenomena and its transient behaviour. Various types of models have been proposed, but the main part of them is rather complicated models expressed by the Park's transform[2].

**Initiation of the self-excitation process** Self-excitation of a standalone generator may take place if a sufficient amount of capacitors is connected to the generator. In order to initiate the self-excitation process, the residual flux in the rotor iron has to be high enough. The residual flux will induce a voltage in the stator when the generator is accelerated to a certain speed. By connecting capacitors to the terminals of the generator, the induced stator voltage will cause a flow of current from the stator [3].

For a given capacitor, an SEIG running at no load requires only a minimum speed for the self-excitation to initiate [4].

**Voltage build-up in the generator** Once the process of self-excitation is initiated, the generator voltage builds up. The voltage build up can more easily be understood by looking at the phasor diagram in Figure 1.

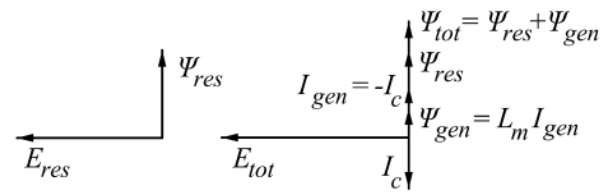


Figure 1: Phasor diagram before and after the self-excitation is initiated [3]

In this figure it can be observed that a current,  $I_c$ , starts to flow from the capacitors once the self-excitation is initiated. This current generates a flux,  $\Psi_{gen}$ , into the generator, with the same direction as the residual flux,  $\Psi_{res}$ . Therefore, the current,  $I_{gen}$ , circulating in the stator reinforces the total flux,  $\Psi_{total}$ . This reinforced total flux causes an even higher stator voltage leading to successive increase in current and flux [3].

Figure 2 shows the generator magnetising characteristic and capacitance for three different frequencies, where the machine magnetising characteristic is simplified by linear segments with a knee point.

For a given capacitance and generator saturation characteristic, the intersection of the capacitance line and the *V-I-curve* of the generator moves as the frequency increases. The voltage build-up comes to halt when the non-linear magnetisation curve for the generator intersects the capacitor voltage curve [1]. This point is the steady state operating point for an induction generator running at no-load with capacitors connected. The no-load steady state operating point is determined by the non-linear magnetisation curve of the generator, the value of the capacitors, and the speed of the generator.

Figure 2 shows that by increasing the frequency, the generator curve is moved upwards, while the slope of the capacitor curve decreases, which results in an increase of steady state operation voltage. This states that connection of capacitors supplying a no-loaded induction generator with a larger reactive power than needed may cause over-voltage at the generator terminals [3].

The intersection point between the saturation characteristic and the capacitor line can be defined in terms of the electrical frequency [1]:

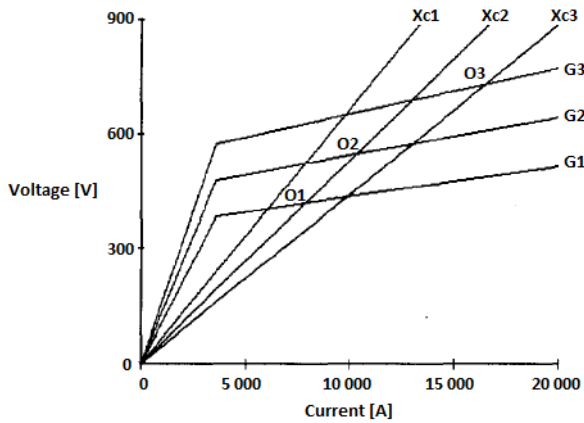


Figure 2: V-I curves for induction generator and capacitor at different frequencies [11]

$$V(\omega) = \frac{\omega A}{1 - \omega^2 L_d C} \quad (1)$$

$$I(\omega) = \frac{\omega^2 A C}{1 - \omega^2 L_d C} \quad (2)$$

Where  $L_d$  is the non-linear magnetisation inductance defined as  $d\Psi/dI$  on the saturated portion of the no-load  $\Psi - I$  curve for the generator, and  $A$  is the interception of the dynamic inductance line with the ordinate, defined as  $\Psi(I) = A + L_d I$ .

In order to achieve a steady state operation point at any frequency, the capacitance must satisfy the following expression:

$$\frac{L_a}{\omega^2 L_d} > C > \frac{1}{\omega^2 L_a}$$

Where  $L_a$  is the inductance defined by the air-gap line of the generator. [1]

## 2.2 Synchronous generators

Three phase synchronous generators are the primary source for all the electric energy produced in a power system. One of the reasons is that the synchronous generator gives the opportunity to decide whether it is desirable to produce or consume reactive power, which gives us the ability to regulate the voltage and power flow in an interconnected grid [5].

The rotor contains a field winding which is supplied by a DC source. This voltage results in a field current,  $I_x$ , which produce the rotor field in the air-gap between the rotor and stator. Controlling the rotor current and hence the rotor produced field, makes

it possible to regulate the induced emf and the reactive power of the generator.

## 2.3 Transmission lines

Small hydro power stations are often connected to a local distribution grid. This grid is usually owned by the local utility company, and is normally operated at a voltage level between 11 kV and 22 kV.

A distribution grid is normally composed of a combination of overhead lines and underground cables. The overhead lines are used for long distances and rural areas, while underground cables are used in urban areas and for underwater crossings. An underground cable is 10 to 15 times more expensive than an overhead line, and it is therefore only used in situations where overhead lines are unsuitable.

From a mathematical point of view, an underground cable can be modelled in exactly the same way as an overhead line. Here, the values of the electrical parameters are the only difference between them. In a cable, the shunt capacitance is strongly dependent on whether the three-phase conductors are screened or not, and on whether the three conductors constitute separate three-phase cables or one common cable [6].

The typical per unit length series inductance,  $L$ , of a cable is about half the inductance of a similar rated overhead line. On the other hand, the per unit length charging current is about 30 times more than for a similar rated overhead line. For a critically long cable, the charging current can be equal to the maximum current of the cable, there will then be no capacity left for transmission of power.

## 3 Over-voltage phenomena in Grunnåi

On the 27<sup>th</sup> July in 2011, several unwanted events took place in Grunnåi power station. By looking at the damages it could be seen that significant over-voltages had occurred in the 22 kV busbars of the power station.

### 3.1 Damages from the events

On the end termination of one of the incoming supply cables, a phase to ground fault had occurred. Two pictures of the end termination and its damage are shown in Figure 3. From the figure it can be seen

that there has been heat generation in the end termination.

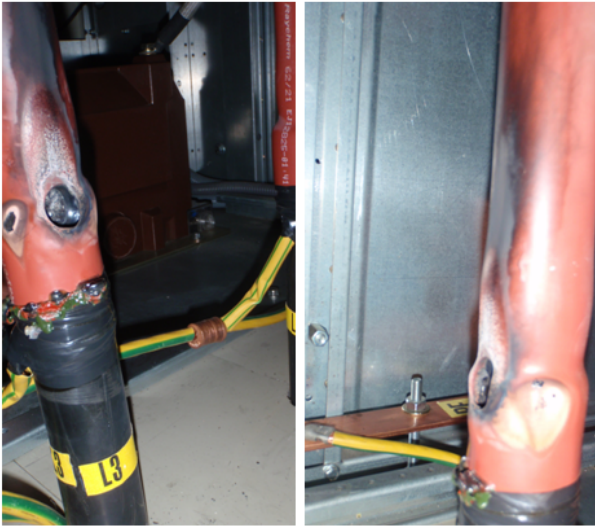


Figure 3: Phase to ground fault on end termination of supply cable

A breakdown of a surge arrester had also occurred under the events. Figure 4 shows the surge arresters with the broken one to the left. By looking at the broken surge arrester and the signs of heat, it is natural to assume that the amount of energy dissipated in the surge arrester was higher than its energy handling capability.



Figure 4: Broken surge arrester to the far-left

Signs of high temperatures and arcs were also seen other places in the rack of the 22 kV busbar, which could be signs of a possible short-circuit due to a high voltage.

Damages did also occur in other places of the radial this day, which indicates that the phenomenon did not only take place locally in *Grunnåi*.

### 3.2 The sequence of events

By looking into the logs of the protection relays in *Grunnåi* and *Seljord* sub-station, it was discovered that several functionalities in each of the protection relays had started to account for triggering. Unfortunately the clocks in the protection relays were not synchronised, so it is impossible to determine which protection relay triggered first.

It is most likely to think that the whole sequence started with a breakdown in the end termination of the incoming cable in *Grunnåi*. The breakdown was most likely caused by a weakness in the end termination due an installation failure. This error may have caused a bad connection or a weakness in the insulation which led to heat generation and degradation of the insulation over longer period of time.

The presumed sequence of events was:

1. Earth circuit fault in *Grunnåi* due to a breakdown in the end termination of a incoming supply cables.
2. The protection relay in the *Seljord* sub-station detected the phase to ground fault and disconnected the *Lønnestad* radial from the rest of the grid in *Seljord* momentarily after the fault was detected.
3. The disconnection resulted in heavy imbalance of active power and reactive power in the now islanded radial.
4. The frequency in the grid increased rapidly since *Grunnåi* did not correct for the overproduction in the island.
5. The generator circuit breaker disconnected *Grunnåi* from the grid, due to triggering of the over-frequency relay (51 Hz and 0.1 seconds). This left the asynchronous generators in *Sagbekken 1* and *Sagbekken 2 and 3* alone in the island.
6. Sufficient amount of reactive power in the grid to initiate self-excitation. The self-excitation led to a successive voltage build-up, which resulted in significant over-voltages in the grid.
7. The high voltage led to a large voltage drop across the surge arresters in *Grunnåi*. The amount of energy dissipated in the surge arrester was higher than its energy handling capability. This caused one of the surge arresters to breakdown.
8. The high voltage exceeded the dielectric strength of air inside the rack of the busbars. The air became ionised and arcing occurred inside the rack. This arcing led to low impedance in the grid, and the process of self-excitation came to halt.

This is the presumed sequence of events, based on the damages and grid configuration. Following chapter presents simulations from the *Lønnestad* radial where different scenarios are carried out.

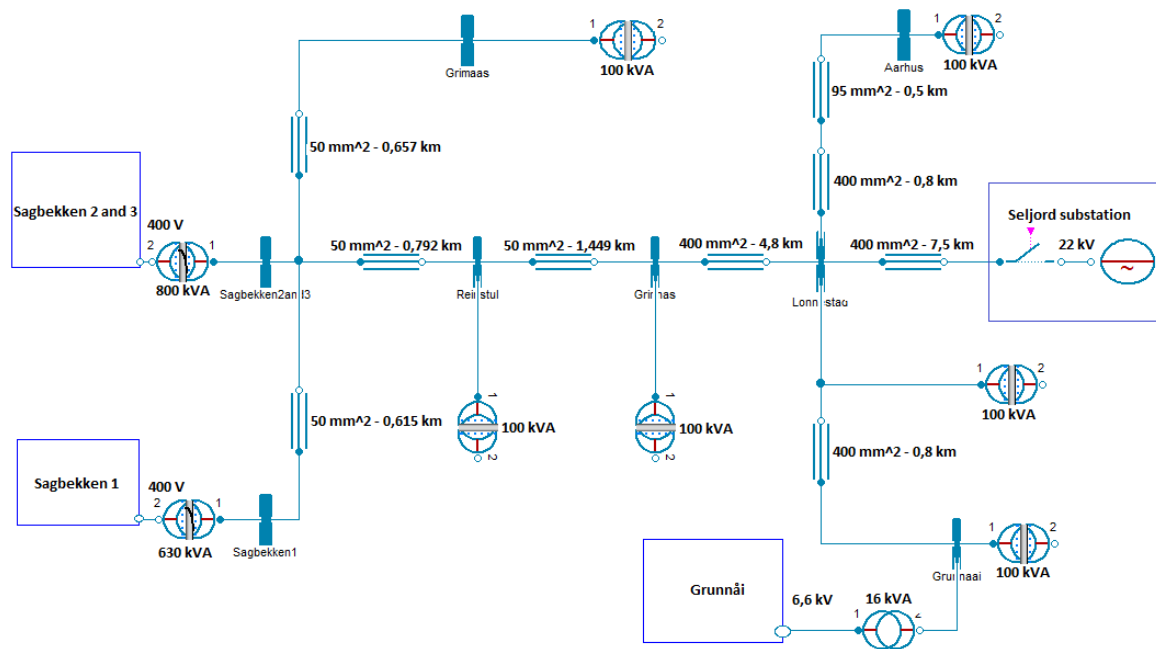


Figure 5: Overview of the Lønnestad radial

## 4 Description of the grid

For investigation of the over-voltage phenomena in *Grunnåi* power station, only the *Lønnestad* radial is of interest. This is the 22 kV radial where the *Grunnåi* power station and *Sagbekken* power stations are connected. The radial is mainly built up with cables, where the total length of 22 kV cables is 17.9 km. An overview of the *Lønnestad* radial is shown in Figure 5. The total power consumption in the radial will vary throughout the day, but is assumed to be 250 kW with a power factor equal to 0.96 during the working hours.

### 4.1 *Grunnåi* power station

*Grunnåi* hydro power station is the largest power station in the *Lønnestad* radial with a synchronous generator of 15.06 MW. The turbine is governed with an infinite droop control, which means that the power station runs at a constant power set-point independently of the electrical frequency in the grid. Figure 6 shows a picture of the synchronous generator inside the power station.

### 4.2 *Sagbekken* power stations

*Sagbekken 1* and *Sagbekken 2 and 3* are two micro power stations that utilise the water from the same river. Both of the power stations are equipped

Figure 6: *Grunnåi* power station with its generator

with asynchronous generators with squirrel cage rotor, where *Sagbekken 1* has three units with a total installed capacity of 400 kW, and *Sagbekken 2 and 3* has four units with a total installed capacity of 475 kW.

Each of the power stations is equipped with protection relays that operate a common generator circuit breaker in each station. Figure 7 shows the four generator units in *Sagbekken 2 and 3*.



Figure 7: The four generator units in *Sagbekken 2 and 3*

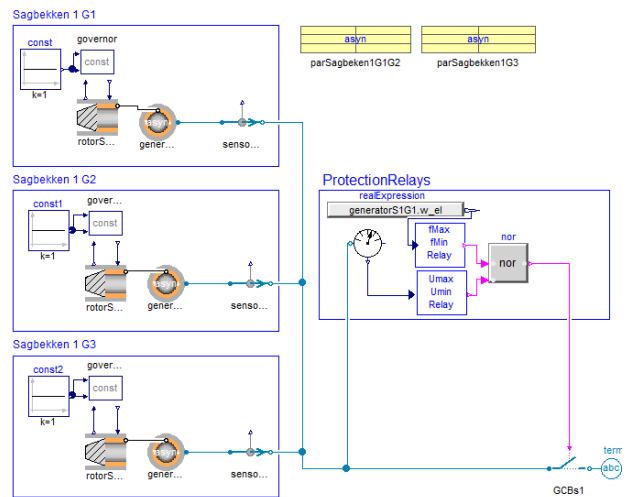


Figure 8: Sub-model *Sagbekken 1*

## 5 Power system simulations

The power system of the *Lønnestad* radial was modelled and simulated using Modelica [7] as modelling language and Dymola[8] as simulation tool. Several models of the power system were created to simulate different scenarios. A small project library was built for the power system, containing the different power system models, main components, and subsystems developed for the models.

The power systems model developed for the *Lønnestad* radial are based on the Electric Power Library[9] which is a commercial library developed by the Swedish company Modelon[10]. The library gives the opportunity to model, simulate, and analyse electric power systems, including AC three phase systems, AC one phase systems, and DC systems. The models can be used for both steady state and transient mode for simulation and initialisation.

Some components like the protection relays and RMS voltage sensors were not available in the Electric Power Library and were therefore created using the Modelica Standard Library [11].

The following sections will give you a brief introduction to the models used and the investigated effects. A more complete and thorough documentation is also available in [12].

### 5.1 Simulation models

Each of the power-station models consist of model for the generator-turbine unit and a protection relay unit. Figure 8 shows an example how the different power stations are build up in principle.

### 5.2 Investigation of the self-excitation process

To be able to investigate the process of self-excitation with different grid configurations, the model in Figure 9 was created.

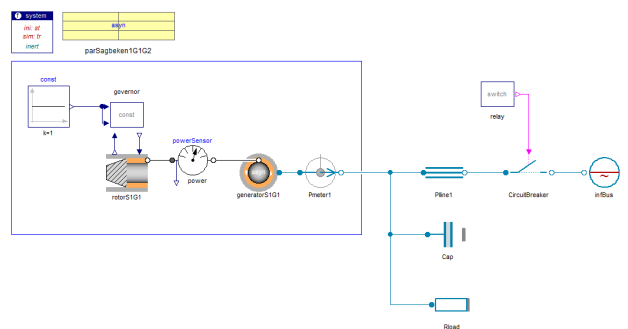


Figure 9: Model used for investigation of the self-excitation process

The model consists of a 100 kW asynchronous production unit and an equivalent distribution line which is connected to an infinite grid. The investigation was carried out as sensitivity analysis where different loads and capacitors were compared in order to determine their impact on the system. At  $t = 1 \text{ second}$  the circuit breaker was opened, which brought the power plant into islanded grid operation with a given capacitive power and resistive load connected.

### 5.2.1 Self-excited induction generator without capacitors

Figure 10 shows how the SEIG behaves when it is suddenly brought into islanded operation without capacitors connected to the generator terminals. Before disconnection it can be observed that the generator operates in steady state where it produces  $98.4kW$  with active power, and consumes  $67.1kvar$  with reactive power. As the induction machine no longer is able to produce the main air-gap and winding leakage flux after the disconnection, it can be seen how the voltage reaches zero when the machine de-excites. Due to the constant mechanical torque,  $T_m$ , on the rotor, it can be seen in Figure 10 how the angular velocity,  $\omega_r$ , increases when the electrical torque,  $T_e$ , disappears:

$$\frac{d\omega_r}{dt} = \frac{T_m - T_e}{J}$$

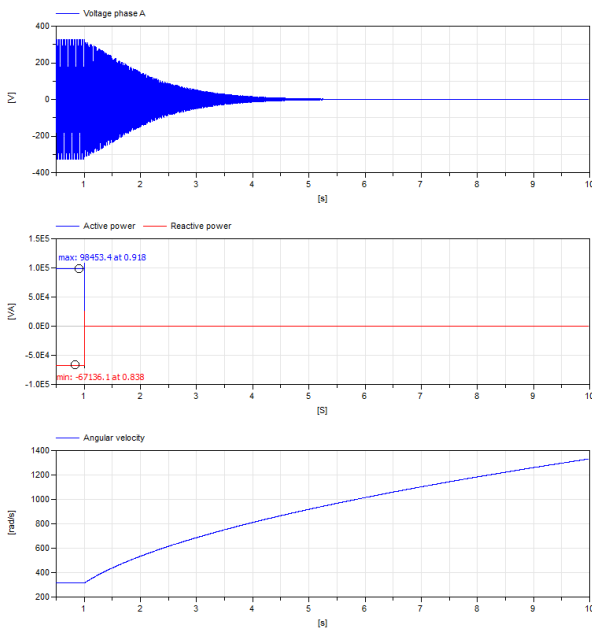


Figure 10: SEIG with no capacitor, disconnection at  $t = 1s$

### 5.2.2 Self-excited induction generator with capacitors

The unloaded SEIG's behaviour with different capacitors after a disconnection is shown in Figure 11. From the figure it can be seen that the SEIG needs at least  $10kvar$  of capacitive power in shunt with the generator to initiate a successful voltage build-up after a disconnection. The figure also shows that

a further increase of the capacitive power results in shorter time from the disconnection to the voltage build-up and a lower peak voltage. This is because larger capacitors provide the minimum amount of reactive power required for self-excitation at a lower angular velocity than smaller ones.

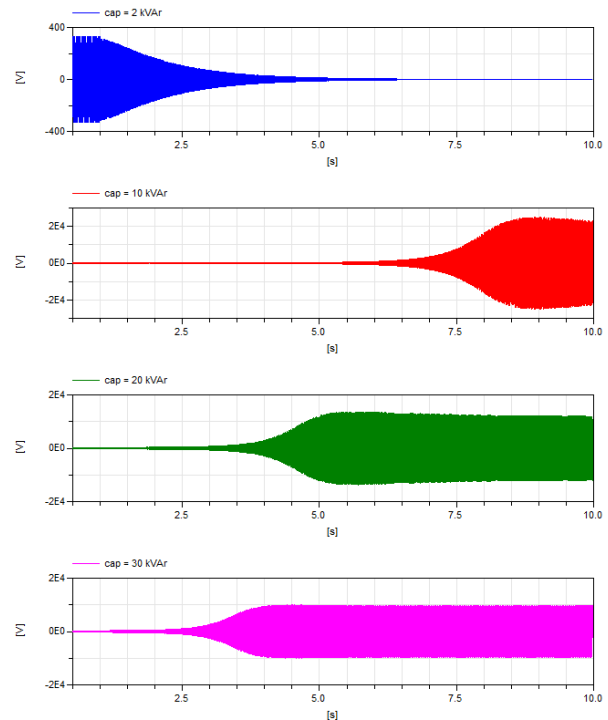


Figure 11: SEIG voltage with different capacitors, disconnection at  $t = 1s$

### 5.2.3 Self-excited induction generator with capacitors and load

To simulate a resistive load's impact on the SEIG, a sensitivity analysis was performed with different resistive loads. For all the different loading scenarios, a  $30kvar$  capacitor bank was connected in shunt with the generator terminals.

By comparing the  $10kW$  simulation in Figure 12 with the similar no-load simulation in Figure 11, it can be observed that the load has a considerable influence on the new operating point. The  $10kW$  load reduces the maximum over-voltage of the  $100kW$  generator from  $10kV$  to  $1.4kV$ .

For the simulation with the  $90kW$  load, it can be observed that the load is too large to initiate the voltage build-up immediately after the disconnection. As the rotor accelerates, it can be seen that the self-excitation is initiated after  $t = 2.5seconds$  when

the angular velocity is large enough for the capacitors.

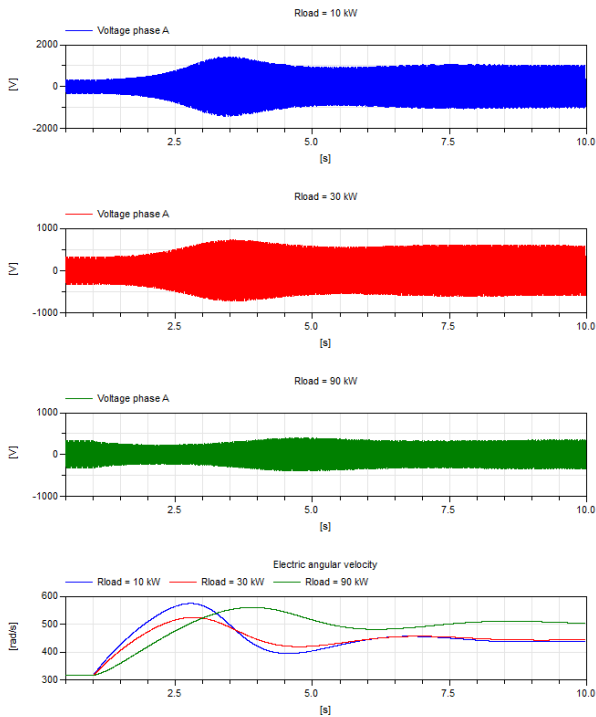


Figure 12: SEIG with 30kvar capacitors and different loads, disconnection at  $t = 1$  s

### 5.3 Simulation with phase to ground fault at Grunnåi

In order to investigate the over-voltage that occurred at the 22 kV busbar in Grunnåi on the 27<sup>th</sup> July 2011, the simulation model in Figure 13 was created. At  $t = 1$  second a phase to ground fault occurs at phase A. Simultaneously as the phase to ground fault occurs, the circuit breaker in Seljord substation disconnects the Lønnestad radial from the rest of the grid due to momentary triggering settings in the protection relay.

Figure 14 shows how the voltages changes when the phase to ground fault occurs at phase A, causing the system to go from a balanced system to an unbalanced system. By looking at the figure it can be clearly seen that the radial has enough capacitive power and low enough load to initiate self-excitation process in the Sagbekken power stations once the Grunnåi power station is disconnected.

Figure 15 shows that it took 1.5 seconds from the phase to ground fault occurring at the Grunnåi power station before all the generators in the radial were dis-

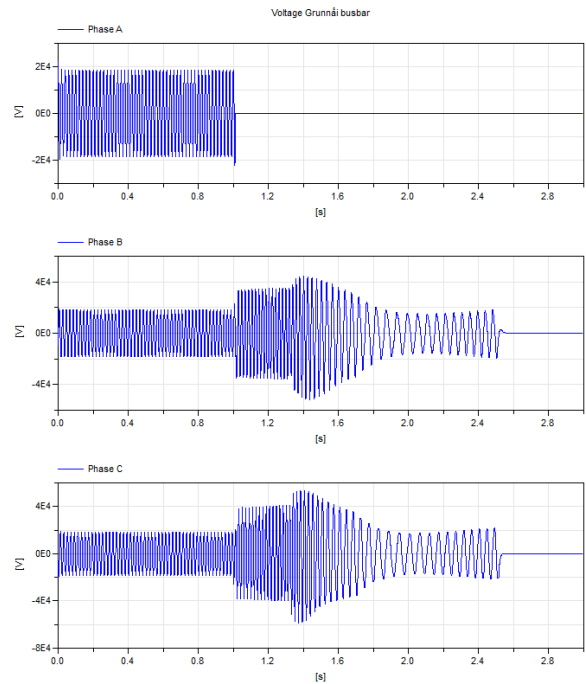


Figure 14: Voltage at Grunnåi busbar, phase to ground fault at  $T = 1$  s

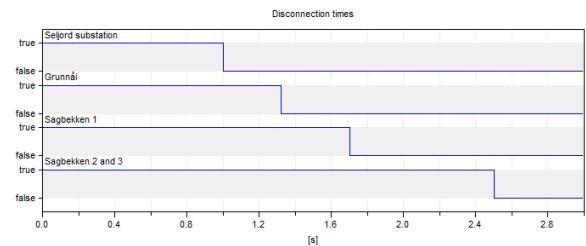


Figure 15: Disconnection times, phase to ground fault at  $T = 1$  s

connected. During this time the over-voltage reached its maximum value of 53.36 kV and was continuously over 30 kV for 0.7 seconds at phase C.

## 6 Discussion

The investigation of the self-excitation process shows how the dynamics of the generator changes when different capacitors and loads are connected to the generator terminals. For a generator running at no load with capacitors connected to the terminals, there exists a minimum speed for self-excitation to occur. If the capacitors do not provide sufficient excitation to initiate the self-excitation at the given speed, the loss of the utility grid causes a sudden in-



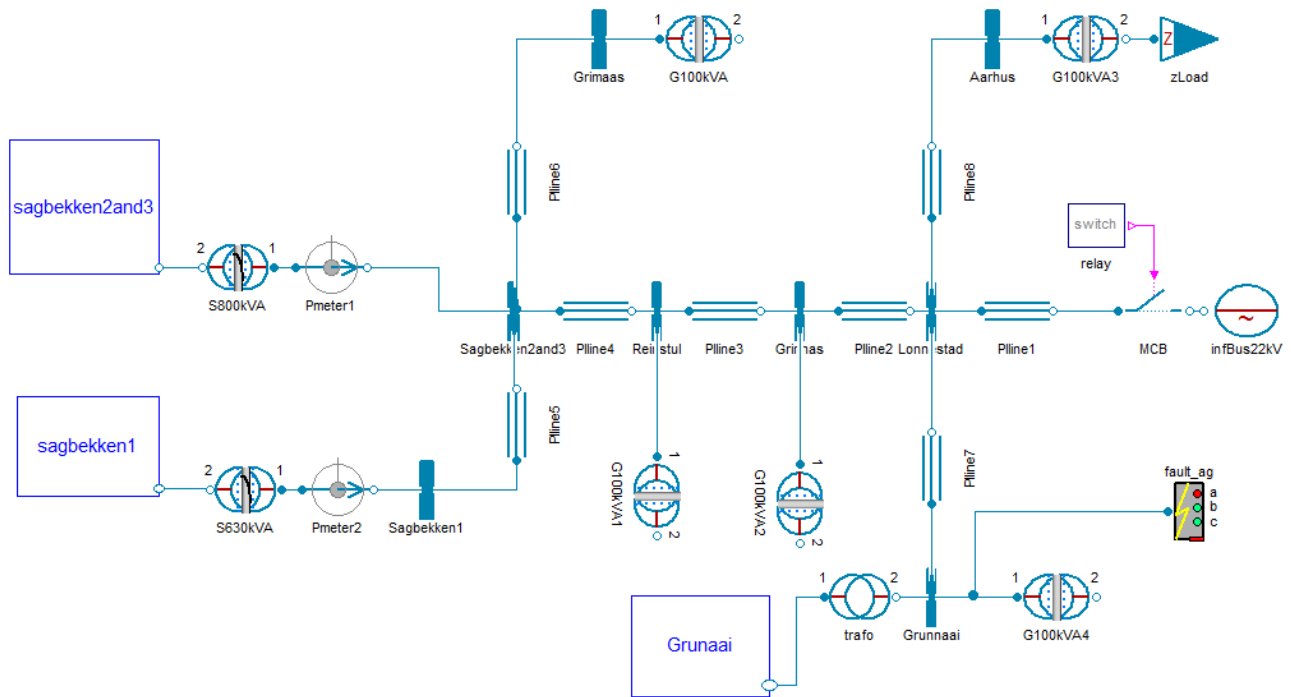


Figure 13: Model of *Lønnestad* radial with phase to ground fault

crease in the slope of the equivalent capacitor line seen by the generator [1]. This change may cause a complete or partial loss of the generator excitation. Due to this, the generator fails to produce an electromagnetic torque large enough to overcome the mechanical torque, which results in acceleration of the rotor.

Unless the residual flux is lost, the self-excitation can be initiated when the angular velocity has increased such that the reactive power from the capacitors is sufficient. Once the process of self-excitation is initiated, the terminal voltage starts to build up. The generator reaches its new stable operating point when the dynamic magnetisation line of the generator intersects the linear capacitor line.

For asynchronous generators connected to a utility grid with much reactive power in form of capacitors or cables, it is crucial with quick detection and low trigger time for over-frequencies to avoid unwanted over-voltages.

The simulations shows that the reactive power in the grid is great enough initiate self-excitation that results in harmful over-voltages, independently of whether the load is connected or not. It is seen that the voltage build-up happens quickly. It takes below 0.4 seconds from the radial is brought into islanded operation to the voltage reaches its peak value.

Significant over-voltages can also occur when a

phase to ground fault arise at *Grunnåi* busbar. Regardless of whether *Grunnåi* power station is connected or a phase to ground fault arise, the self-excitation of the *Sagbekken* stations will result in harmful over-voltages. For all the simulations, the magnitude and length of the over-voltages are greater than the thermal stability limit of the surge arresters in *Grunnåi* power station.

Correct parameters for the protection relays are essential for providing sufficient protection of the grid. This is also the easiest way to protect the *Lønnestad* radial against harmful over-voltages. Simulations show that the peak value of the angular velocity can vary dependent on the load scenario. Correct parameters for detection of over-voltages are therefore most important for the protection relays.

## 7 Conclusion

This paper investigates the system dynamics in the *Lønnestad* radial when it is brought into islanded operation. Modelling and simulation of the transient behaviour of an asynchronous generator is a fairly complex task that requires good knowledge of electric machinery and dynamic systems. Due to this, there is often a lack of knowledge in small utility companies when it comes to the asynchronous gen-

erator.

The asynchronous generator has the opportunity to operate as a standalone unit if the amount of reactive power in the cables or capacitors is sufficient. For the *Lønnestad* radial, it was proven that the amount of reactive power in the grid is large enough to initiate self-excitation of all the seven asynchronous generators in the radial.

The self-excitation leads to fast voltage build-ups that results in a harmful over-voltage in the distribution grid. For the simulations with load connected, it was observed that the over-voltage in the distribution grid reached its maximum voltage of circa 50 kV, only 0.4 seconds after the radial was brought into islanded operation.

This is a type of over-voltage that requires a great deal of knowledge regarding self-excitation to ensure good protection of the grid. Normal protection methods as surge arresters will not be adequate, since these are designed to protect against surge voltages, and not transient over-voltages with several seconds duration. Simulations performed in this paper show that it is crucial with correct protection parameters in the *Sagbekken* stations to protect equipment in the *Lønnestad* radial against over-voltages caused by the generators. To avoid unwanted voltage build-ups, correct parameters for over-voltage detection is the most important protection. It is recommended to have momentarily disconnection when the voltage exceeds a given value.

## 8 Acknowledgements

The work was carried out as part of the Master thesis of Håkon Molland Edvardsen [12] in cooperation with Skagerak Energi, Porsgrunn, Norway and under supervision of Dietmar Winkler.

## References

- [1] L. Tang and R. Zavadil. “Shunt capacitor failures due to windfarm induction generator self-excitation phenomenon”. In: *IEEE Transactions on Energy Conversion* 8.3 (1993), pp. 513–519. DOI: 10.1109/60.257067. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=257067>.
- [2] Robert H. Park. “Two-reaction theory of synchronous machines generalized method of analysis-part I”. In: *Transactions of the American Institute of Electrical Engineers* 48.3 (1929), pp. 716–727. DOI: 10.1109/T-AIEE.1929.5055275. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5055275>.
- [3] F. Sulla. *Island Operation with Induction Generators: Fault Analysis and Protection*. Department of Measurement Technology and Industrial Electrical Engineering, Lund University, 2009. ISBN: 9789188934512. URL: <http://books.google.no/books?id=PiQbtwAACAAJ>.
- [4] Dawit Seyoum, C Grantham and F Rahman. “Analysis of an isolated self-excited induction generator driven by a variable speed prime mover”. In: *Proc. AUPEC*. Vol. 1. 2001, pp. 49–54.
- [5] Ned Mohan. *Electric Power Systems: A First Course (Coursesmart)*. Wiley, 2012. ISBN: 1118074793.
- [6] Jan Machowski, Janusz Bialek and Dr Jim Bumby. *Power System Dynamics and Stability*. Wiley, 1997. ISBN: 0471956430.
- [7] Modelica Association. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification*. Version 3.2. 2012.
- [8] Dassault Systèmes. *Dymola*. 2013. URL: <http://www.dymola.com>.
- [9] Modelon. *Electric Power Library*. 2013. URL: <http://www.modelon.com/products/modelica-libraries/electrical-power-library/>.
- [10] Modelon. *Homepage*. URL: <http://www.modelon.com>.
- [11] Modelica Association. *Modelica - Free library from the Modelica Association*. 2010. URL: <https://github.com/modelica/Modelica>.
- [12] Håkon Molland Edvardsen. “System dynamics of asynchronous generators at islanded grid operation”. MA thesis. Telemark University College, 2013. DOI: 2282/2125. URL: <https://teora.hit.no/handle/2282/2125>.

# Hybrid Energy System Modeling in Modelica

William R. Binder<sup>1</sup>   Christiaan J. J. Paredis<sup>1</sup>   Humberto E. Garcia<sup>2</sup>

<sup>1</sup>Georgia Institute of Technology  
813 Ferst Drive, Atlanta, GA 30332, USA  
binderw@gatech.edu, chris.paredis@me.gatech.edu

<sup>2</sup>Idaho National Laboratory  
2525 N. Fremont Drive, Idaho Falls, ID 83415-3675, USA  
Humberto.Garcia@inl.gov

## Abstract

In this paper, a Hybrid Energy System (HES) configuration is modeled in Modelica. Hybrid Energy Systems (HES) have as their defining characteristic the use of one or more energy inputs, combined with the potential for multiple energy outputs. Compared to traditional energy systems, HES provide additional operational flexibility so that high variability in both energy production and consumption levels can be absorbed more effectively. This is particularly important when including renewable energy sources, whose output levels are inherently variable, determined by nature.

The specific HES configuration modeled in this paper include two energy inputs: a nuclear plant, and a series of wind turbines. In addition, the system produces two energy outputs: electricity and synthetic fuel. The models are verified through simulations of the individual components, and the system as a whole. The simulations are performed for a range of component sizes, operating conditions, and control schemes.

*Keywords: hybrid energy system; Modelica; multiple-input; multiple-output; renewable power; optimization*

## 1 Introduction

Over the many years during which the current energy ecosystem was designed and has evolved, technological and social growth occurred slowly and environmental impact was not a primary concern. However in the current context of global climate change and economic volatility, the old energy system is far from optimal. Renewable energy sources promise an alternative that is both low cost and environmentally friendly. However, renewables also pose a challenge: they introduce significant variability in their output.

The output variability and uncontrollability of renewables require grid operators to maintain a larger spinning reserve. If a large plant is asked to reduce generation as the result of using lower-cost solar or wind, the plant has to ramp down and eventually ramp up again. This is harmful, as the cyclical loading reduces the life of the plant, or requires costly maintenance.

In addition to output variability, another concern when designing a power plant is the potential change in the cost of inputs as well as the price of outputs. In a conventional plant, there is only one input and one output. A consequence of this one-to-one energy mapping is that the plant has little control over its profitability when a price change occurs. An increase in the cost of the input, or decrease in the price of the output will most likely cause a decrease in profit. There is a clear tradeoff between using the low cost renewables and the higher level of control that conventional plants offer.

One approach for managing such tradeoffs is the development of Hybrid Energy Systems (HES). HES consider multiple energy inputs and outputs in one system. A Multiple Input Single Output (MISO) HES uses two or more energy inputs, such as nuclear and wind, and produces a single output, most often electricity. A Multiple Input Multiple Output (MIMO) system includes multiple inputs and produces multiple outputs, such as producing both electricity and synthetic fuel. Since they are capable of dynamically utilizing diverse inputs and outputs with different costs, MIMO HES provide a flexible and robust alternative for the energy ecosystem.

There are distinct advantages to having multiple energy inputs and outputs. Consider the HES architecture shown in Figure 1. This MIMO HES uses a renewable energy source (e.g., wind or solar), a non-renewable energy source (e.g., nuclear), and a carbon source to provide both electricity and chemical products. The HES whose models are presented in this

paper use the architecture illustrated in Figure 1. One proposed mode of operation when generation from renewables is high is to direct steam from the non-renewable source to the chemical plant, which requires high-temperature steam to produce synthetic fuel. The amount of steam diverted to the chemical plant can be dynamically varied so that the HES can load-follow, i.e., it can quickly react to changes in the availability of renewables or the demand from the grid. The price of electricity may change dramatically, not only during different seasons, but also during the course of a day. It is even possible that the price of electricity becomes negative, and plants must pay to dispose of the electricity they produce. Producing for a short period at negative prices may sometimes be acceptable because the cost of ramping down and ramping back up is greater than the expected loss from paying to push the electricity onto the grid [5]. Plants or MISO systems that only produce one output have little flexibility as they cannot produce another product. MIMO systems however, are more flexible. If the price for one of its outputs drops significantly, the system can produce other outputs that are still highly priced. This increased diversity allows for plant owners to generate greater and more consistent profits. Since the MIMO system does not need to cycle the non-renewable source as frequently, its performance, reliability, and capacity factor are expected to be larger than for other systems. This in turn results in an opportunity for increased renewable penetration as well as profitability for plant owners.

Although MIMO systems provide many benefits, there are also some disadvantages. Due to the increased complexity of these plants, their design,

analysis, and control becomes more challenging. A second disadvantage concerns the lifespan of key components, such as heat exchangers, which may suffer from additional wear due to thermal cycling. Another concern for HES is that they have to deal with dynamic conditions that cannot be well represented in a static model. In addition, dynamic simulation is critical to controller design and optimization. To support dynamic modeling, models are implemented using the Modelica language, and simulated using Dymola [9].

In the remainder of this paper, other HES models from the literature are first reviewed. Then, models are presented for the thermo-fluid systems, electrical system, and chemical system of the HES architecture shown in Figure 1. Simulations of these models are then reported and interpreted. In addition, lessons learned throughout the model creation process are presented. Finally, concepts for future work are described.

## 2 Related Work

Modeling and optimization of HES is not entirely new. There are multiple examples of research being made in this area [13-15]. [15] is an example of a hybrid wind-solar energy system for small scale applications. The hybrid system does not connect with the electrical grid and therefore avoids any complications associated with having to do so, such as maintaining the same phase angle and voltage. This is another area where HES can be applied; however the systems designed for off-grid use do not help alleviate the non-renewable production of electricity that

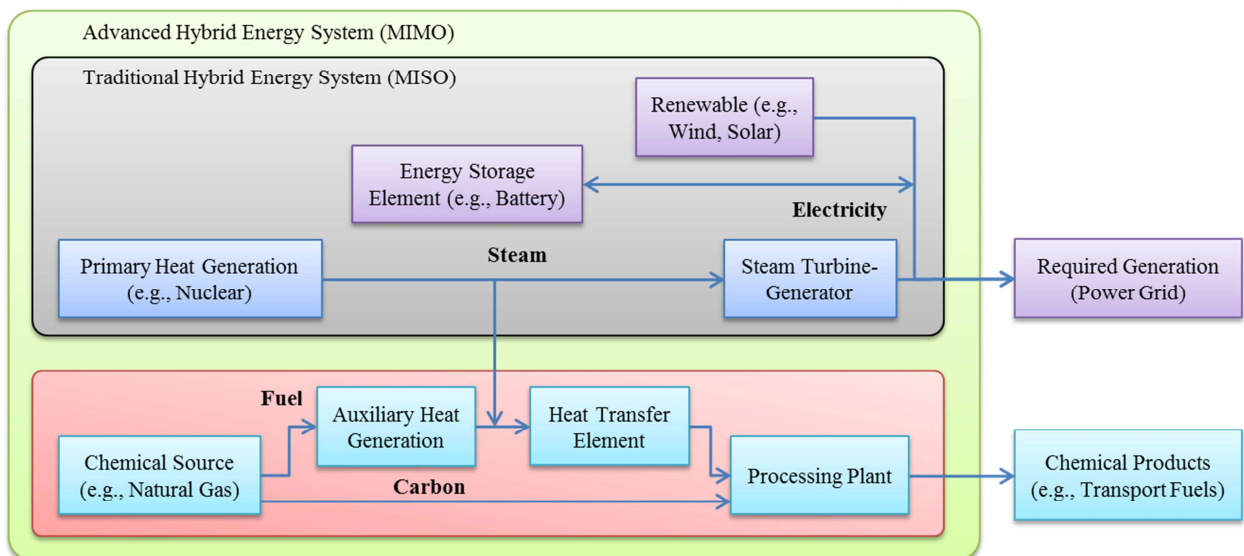


Figure 1: The architecture for an Advanced Hybrid Energy System (MIMO) [6, 7]

dominates the electrical grid. There are examples of on-grid applications, such as [13].

In [13], the task of handling renewable generation for the purpose of connecting to the electrical grid is addressed. This system however, focuses on the control and optimization of a wind power generation plant, as compared to HES. In this case, the plant is optimized to account for dynamic changes that can occur in wind. Even so, this does not eliminate the plants dependency on there being a sufficient amount of wind to power the electrical grid. The issue of connecting a HES to the grid is discussed in [14].

In the case of [14], the HES consists of a photovoltaic, diesel, battery combination. This is used for the purpose of supplying electricity to rural Saudi Arabia. In this application, software called the Hybrid Optimization Model for Electric Renewables (HOMER) was used to evaluate and optimize the HES [12]. HOMER allows for the design of HES architectures, in a high level view. It can be used to evaluate the basic structure of HES, but does not deal with the dynamic events occurring within a system. In addition, HOMER handles loads such as electricity and thermal, abstracting the smaller components that would actually be involved in the design of a large scale plant.

There appears to be a gap in terms of designing a full scale HES plant for the purpose of connecting to the electrical grid. The previous examples show that there is potential for economic and environmental improvements from creating HES configurations, either new or from converting old plants.

### 3 Models

The modeling of the HES described above was divided into five main sections. The five sections are: the nuclear reactor, two steam cycles, a chemical plant, and the electrical component. These subsystems are shown in Figure 2, which shows the individual models connected to illustrate the final structure of the HES model. The nuclear reactor supplies the primary heat generation that is utilized in the first steam cycle. The steam cycles consist of one segment that extracts work from the heated steam for the purpose of generating electricity, and the other superheats steam for the chemical plant. The chemical plant takes hot steam and natural gas to produce synthetic fuel. The electrical section contains the renewable source, in this case wind turbines, with a battery and connects with the electricity generated elsewhere to power the electrical grid. The models

used are from the Modelica basic library as well as the ThermoPower library, which is used for modeling of the thermo-fluid components [1].

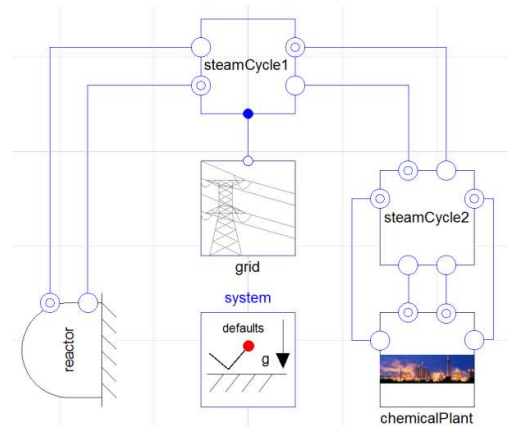


Figure 2: Top-level model for the HES

#### 3.1 Nuclear Reactor

The nuclear reactor's purpose is to supply heat to the steam cycle. This is the largest energy input to the HES. The model for the reactor is shown in Figure 3. The model is simplified as the primary concerns are with the dynamics of the interconnected system involving the electrical and chemical components. The reactor uses a heat source that ramps from a nominal starting power to its full load. This represents a plant powering up from reduced load. This allows for a system that uses the reactor to warm up to steady state operating conditions, as compared to trying to start all of the equipment at full load. This heat transfer is applied to a pipe with water flowing through it to represent a main heat exchanger.

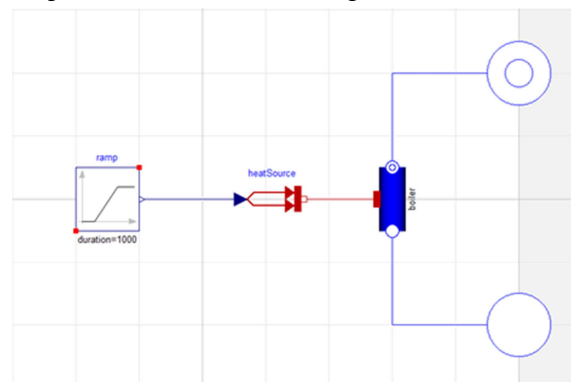


Figure 3: The model for the reactor.

#### 3.2 Steam Cycle

The steam cycle models are responsible for capturing the transfer and distribution of energy of the thermo-

fluid portion of the HES. This is broken into two segments: one whose primary purpose is the production of electricity, and the second being the superheating of steam to the chemical plant. The distinction of primary purpose is necessary due to the fact that both segments produce electricity and heat water.

The portion of the steam cycle primarily responsible for generating electricity is shown in Figure 4. There are some critical observations to note about this model. Since the HES can vary the amount of steam being utilized to produce electricity, if just one turbine were to be used there would be times where it would not be fully utilized. For this reason, multiple turbines are present for use in a cascading fashion, as in, at low power requirements, only the “60% Turbine” may be used. As the electrical demand in-

creases, the “30% Turbine” and eventually the “15% Turbine” will also be added. Other than this, the model represents a Rankine cycle with safety components as well as a boiler to distribute steam to the chemical plant. When the power demanded from the turbines is lowered by the addition of renewable power, the flow rates through the turbines will decrease. This will cause a corresponding increase in pressure as the same amount of heat is being added to the cycle, regardless of the turbine output. To regulate this pressure, a pressure relief valve is present, but instead of venting this excess energy, the excess steam is used to heat condensate water coming from the chemical plant, thus providing it with steam. A temperature control valve is also included to more precisely control the temperature of the colder water entering the nuclear reactor.

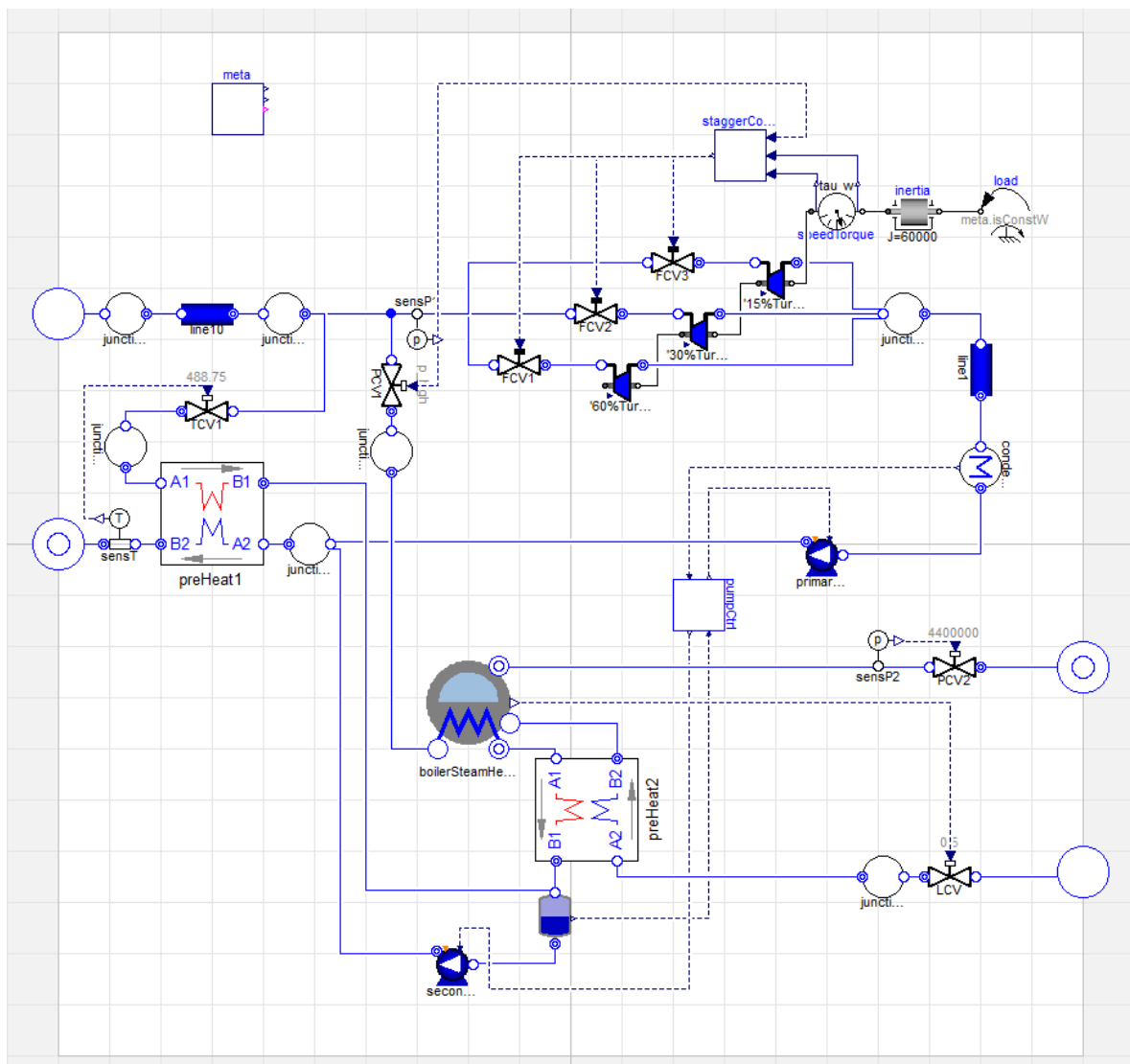


Figure 4: The model for the steam cycle whose primary function is to produce electricity.

The model for the heat exchanger is a simplified one that is capable of simulating phase changes in the system. Other heat exchangers had been tested, such as those from the ThermoPower library and the Modelica standard library, however they did not fit the needs of this application [1, 9]. The complication in using these models is that they were not designed to allow the fluid to undergo a complete phase change. The cooling of steam to water results in significant changes in the fluid properties, such as density. This causes simulation stiffness. Often, at least one of the heat exchangers is not always in use, it is possible for the contents of that heat exchanger to cool to the saturated liquid vapor temperature, and causes corresponding stiffness.

To resolve this, a model was created that utilizes pipes with no volume. Since there is no volume, the large changes in fluid properties that occur throughout the heat exchanger are largely ignored, and only the inlet/outlet conditions of the fluid determine the

heat transfer.

To capture as much detail as reasonable, the heat transfer,  $Q$ , and efficiency of the heat exchanger,  $\eta$ , are calculated in equations (1), (2), (3), and (4) [2, 10]:

$$Q = \Delta h_{Hot} w_{Hot} = -\Delta h_{Cold} w_{Cold} \quad (1)$$

$$\eta = \frac{T_{ColdInlet} - T_{ColdOutlet}}{T_{ColdInlet} - T_{HotInlet}} \quad (2)$$

$$\alpha = k \left( \frac{1}{\dot{m}_{Hot} C_p_{Hot}} + \frac{1}{\dot{m}_{Cold} C_p_{Cold}} \right) \quad (3)$$

$$\frac{T_{HotOutlet} - T_{ColdOutlet}}{T_{HotInlet} - T_{ColdInlet}} = e^{\alpha} \quad (4)$$

where  $\Delta h$  is the change in enthalpy,  $\dot{m}$  is the mass flow rate, and  $C_p$  is the specific heat of the fluid.  $k$  is the assumed heat transfer coefficient. This ignores some of the effects that are present during a phase change in a heat exchanger, however, this occurs

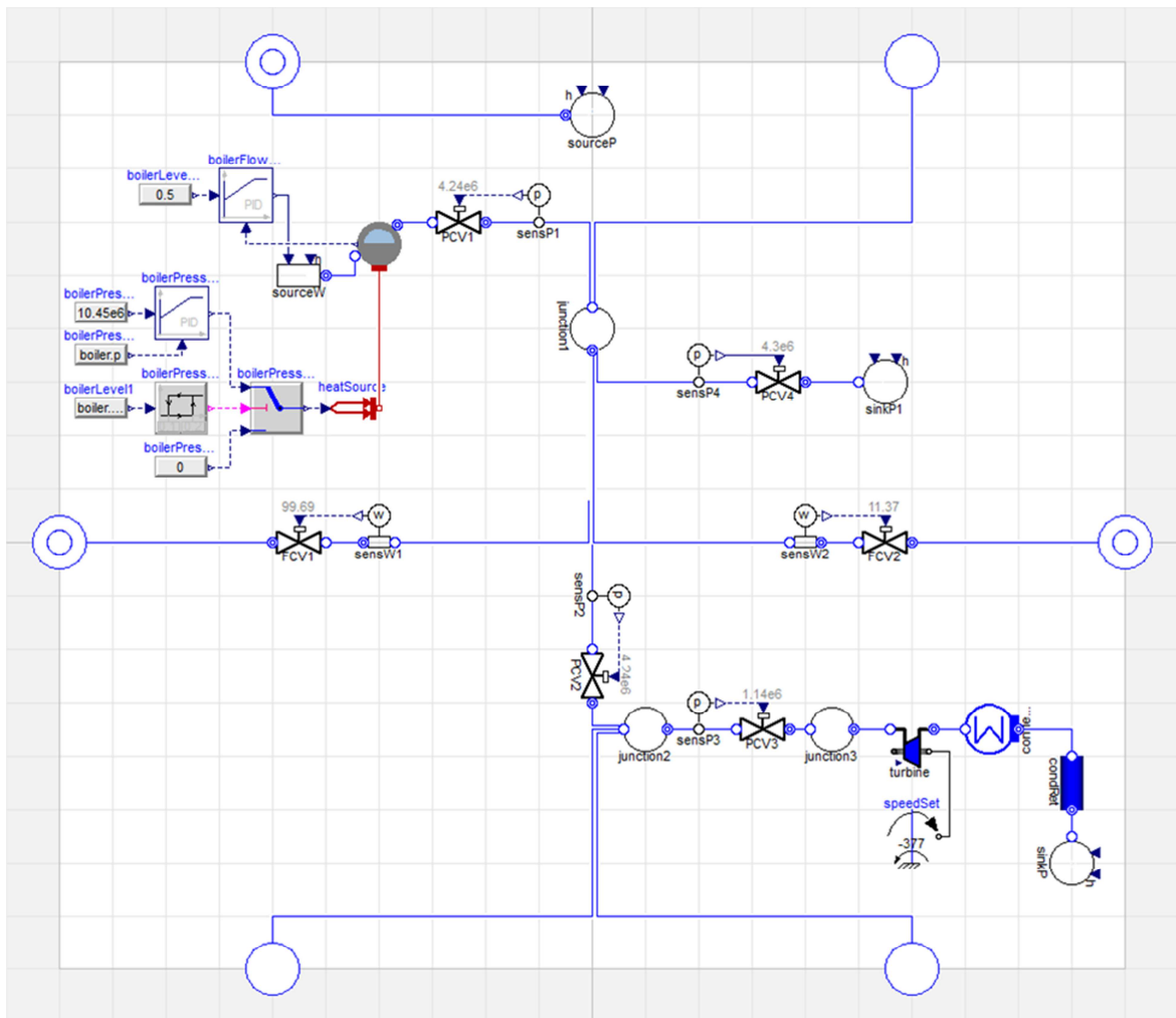


Figure 5: The model for the steam cycle that is primarily responsible for superheating steam to the chemical plant.

under conditions where capturing that change is considered insignificant. A phase change can occur when the flow rate of the hot side is small relative to the cold side. This can occur in the second heat exchanger in Figure 4, labeled preHeat2. It is expected that this condition will not change the results drastically, and the dynamics of the heat exchanger when it is not transferring large amounts of heat are not a concern currently.

Figure 5 presents the other side of the steam cycle, the one responsible for superheating steam to the chemical plant. The purpose of this model is to take steam generated from the model in Figure 4, superheat it, and then transport it to the chemical plant. There are locations where the condensate water addition and removal are represented by sources and sinks of water. This is done for simplicity instead of closing the loop of flowing water. The dynamics of the condensate water used in this system are not considered a significant concern. In addition, the justification for this is that the condensate water is merely being pumped around the system, and the power required to pump liquid water is small relative to the power produced by most Rankine cycles [10].

In the event that there is an insufficient amount of steam being transferred to the model in Figure 5, this system also has auxiliary heat production generate the necessary additional steam. Furthermore, the steam that returns from the chemical plant, in addition to excess steam produced, if any, is put through a turbine to capture energy that would have been

wasted otherwise.

### 3.3 Chemical Plant

The model of the chemical plant is shown in Figure 6. The chemical plant model utilizes a methanol medium, and most of the sub-models perform their functions mathematically, via transfer functions, instead of using the energy based components of the water system. This is done to simplify the overall analysis, as the intricacies of transforming natural gas to gasoline and liquefied petroleum gas (LPG) are not of interest.

### 3.4 Electrical

The electrical model is the location where the renewable source and the electrical grid, which receives power from the HES, are included. The renewable generation for the HES under consideration is a series of wind turbines. The model uses representative data for wind speed for a location in Idaho from the Western Wind dataset, which was made available from NREL (National Renewable Energy Laboratory) [6, 11]. This data assumes a height of one-hundred meters and has wind data for every ten minute period. The year period of 2006 is used for simulations. This wind speed,  $v$ , is then mapped to rotational power,  $P$ , by equation (5) [13]:

$$P = \frac{1}{2} \rho A v^3 C_p \tag{5}$$

where  $\rho$  is the air density,  $A$  is the cross sectional

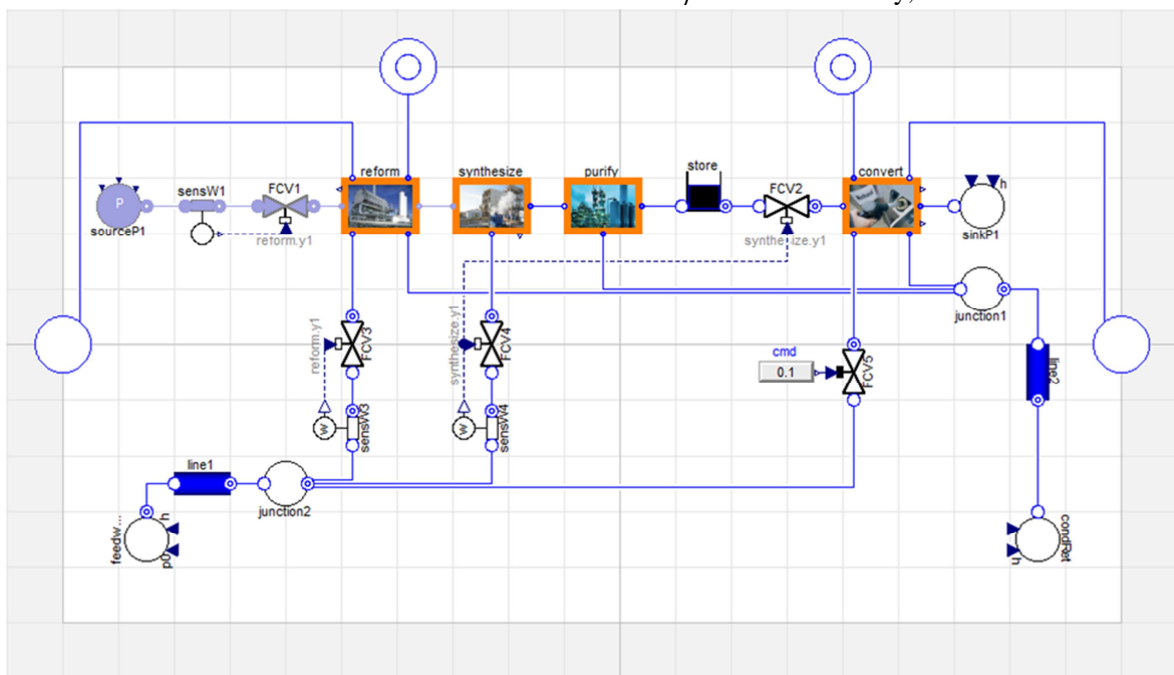


Figure 6: The model for the chemical plant.



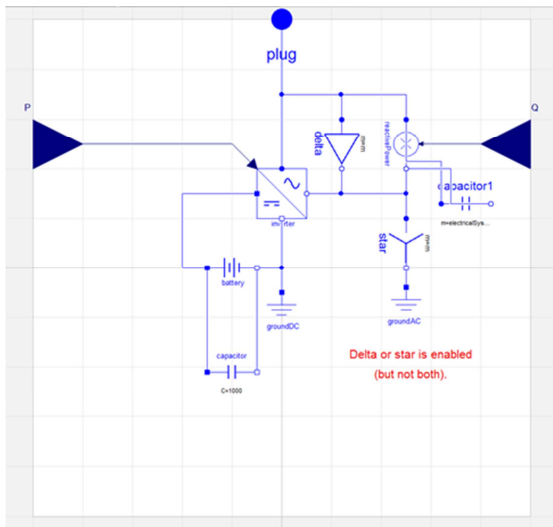


Figure 7: The model for a battery.

area of the blades, and  $C_p$  is the power coefficient. This rotational power is applied to an electrical generator, which outputs three-phase AC. It is worth noting that the models that utilize three-phase AC are done so in dq0-reference frame to improve simulation time.

In addition to the wind turbines, another notable component is the electrical battery. The model for

the batteries is shown in Figure 7. The AC signal interacts with an inverter, which converts AC to DC and DC to AC depending on whether the battery is charging or discharging. The battery itself is a resistor-capacitor unit arranged in series and parallel until the necessary voltage and capacity is reached, also known as the Thevenin Battery Model [3, 8]. The overall electrical model is shown in Figure 8. The plug represents input electrical power generated elsewhere. The two batteries in Figure 8 are controlled by a grid supervisor, which dictates the real and reactive power demanded.

The electrical grid is the location where all of the electrical signals come together to power a model of the U.S. electrical grid. The outputs of the turbines from the steam cycle, as well as output from the wind turbines, each with their own battery to handle transients, are connected to the U.S. grid via a circuit breaker. This circuit breaker connects the HES to the electrical grid when it is closed. In addition, lines and substations are included to represent how the electricity would travel to the grid and include relevant losses from transmission. The lines contain inductor-resistor circuits. Thus, the losses in the resistors, as well as the dynamics from the inductors are taken into account.

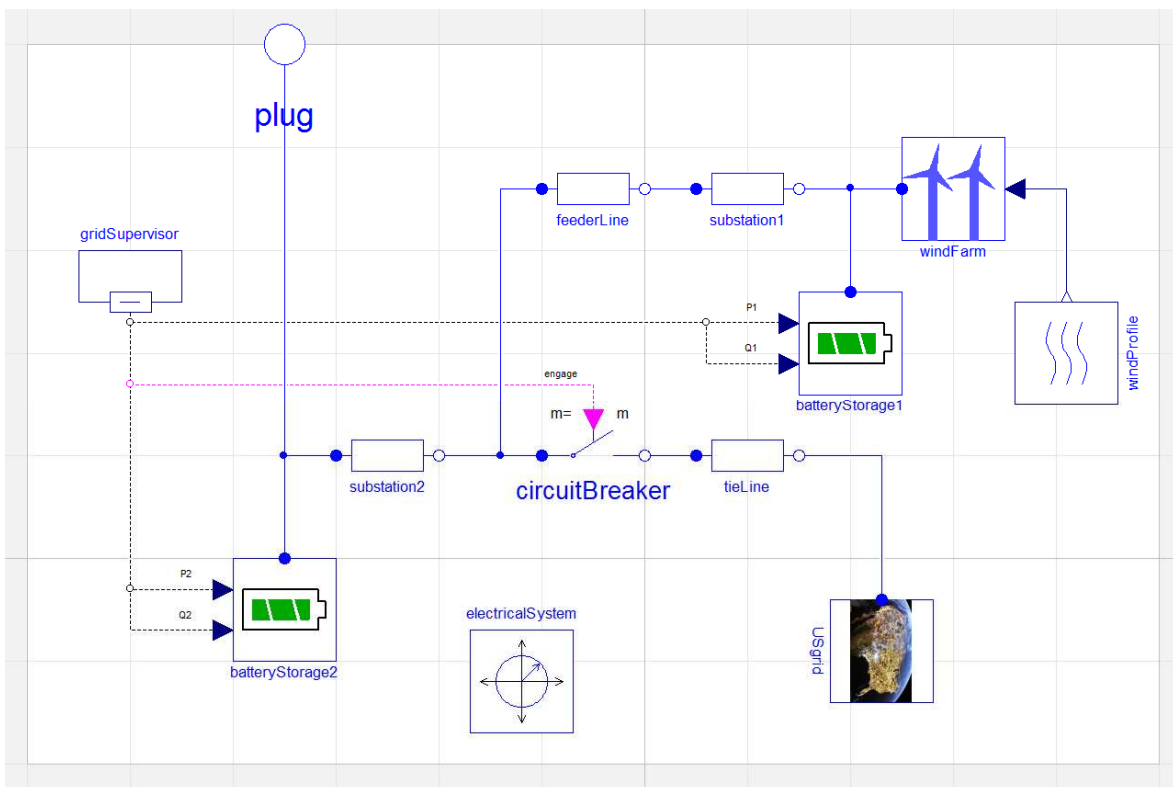


Figure 8: The electrical model, which includes the wind turbines, batteries, and the grid.

## 4 Simulation Results

To verify that the models created represent the system of interest, simulations are conducted. The results of these simulations are analyzed to ensure that the models behave as expected. Before combining the models and simulating the full system, individual components were tested first.

One of the first lessons learned came from the simulations of the first steam cycle. By closing the fluid loop, the simulations immediately became sensitive to various parameter values. Changes in the sizes of components in the loop cause the system to react in the form of large transients that dominate the startup period. These transients cause the simulation to be stiff and result in the simulation time increasing significantly. Attention to initial parameter values is paramount to meaningful results for this situation. This is one reason why the reactor is ramped up to load, as compared to starting at max load.

As mentioned previously, the other heat exchanger's that are considered initially resulted in simulation problems for the first steam cycle. The low flow rate of steam causes the steam to condense to liquid water, which results in a large increase of density, drastically increasing the simulation time. The change in density causes the simulation to both progress slowly, and result in the model not behaving as intended. One result of the large increase in density is that objects with a finite volume began decreasing their pressure rapidly. In some cases, this low pressure causes some components to operate in backflow. For the separator in Figure 4, this result does not make sense. A mixture of liquid water flowing from the top port into a heat exchanger is not realistic or indicative of what would normally be expected for a separator. The phase change also causes problems with the medium model used for water. The changing phase results in the properties, in addition to the density of the water, to change drastically. Since a phase change occurs, the medium model needs to be capable of varying properties, but should do so in a smooth manner so as to avoid stiffness issues.

To explore the interactions between the thermo-fluid systems, a simulation is conducted consisting of only the nuclear reactor, the steam cycles, and the chemical plant, represented in Figure 9. In order to capture the effect of the wind power, the load required from the turbines is reduced as if the wind turbines were connected. Assuming that the overall power that would go to the grid is set to be constant, as the wind turbines produce more power, the turbines accordingly reduce their electrical production.

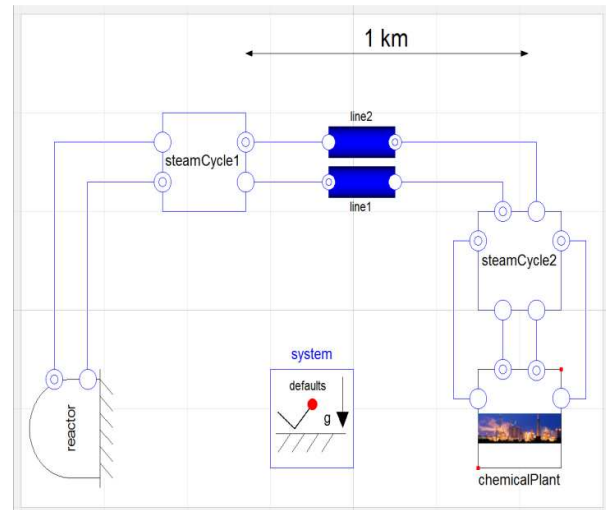


Figure 9: The model containing the reactor, the steam cycles, and chemical plant.

Conversely, as the wind turbines produce less power, more electricity is requested from the turbines.

The simulation is conducted using nominal parameter values that will cause all of the turbines to be used for at least a short period of time. Of particular interest are the flow rates of the steam through the turbines, preheater, and the secondary boiler. The results for this are shown in Figure 10. This simulation in Dymola takes approximately 45 seconds to simulate almost 14 hours of HES operation.

Figure 10 shows the result of the simulation. These results are reasonable for the conditions being simulated. Initially the system is allowed to ramp up and so no power is requested, hence causing large amounts of steam to divert toward the secondary boiler, denoted in pink. As power is requested, the flow rate of steam going to the secondary boiler decreases with a corresponding increase in flow rate through the 60% turbine, denoted in blue on the left. It quickly reaches its operating flow rate and the 30% turbine begins turning on, denoted in red. Around four hours, even the 15% turbine is at its operating flow rate. The wind turbines, based on the wind speed, begin producing significant power at four and a half hours, reducing the load requested from the turbines, and causing the 15% and 30% turbines to turn off as more steam goes toward the secondary boiler, as expected.

The simulation also catches a significant transient move close to twelve hours into the simulation. This is likely related to the low power being demanded from the turbines, with significant quantities of steam being sent to the secondary boiler. The temperature of the condensate water that would inlet to the reactor drops, causing the preheater to turn on.

The cause of the temperature drop may be related to the temperatures in the second heat exchanger that connects to the secondary boiler.

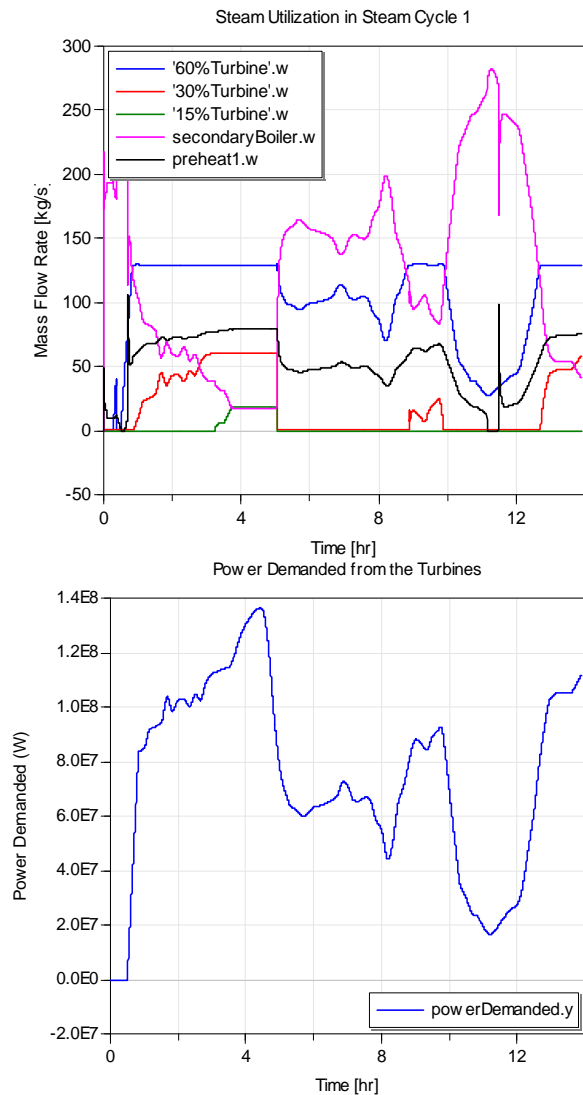


Figure 10: The results of simulation for the flow rates of the turbines, secondary boiler, and preheater1 above. Below shows the power demanded from the turbines.

## 5 Summary and Future Work

In summary, models for a HES that utilizes a non-renewable source (nuclear), a renewable source (wind), and a chemical plant are presented. This work shows that Modelica can be effectively used to model large complex systems, and still allow simulations of dynamic conditions. In addition to modeling

the system, Modelica offers a significant advantage for evaluating control algorithms, which will likely influence the success of HES. The application of this model to determining the optimal design and control schemes will be the target of future work.

Ultimately, the economic and environmental benefits of the proposed MIMO HES described above are to be estimated in order to be optimized. The controllers and other variables, such as the quantity of wind turbines, will be optimized to maximize the profit of the HES to a plant owner. To quantify the environmental benefits of the HES so that the HES can be optimized using profit, a price per ton of CO<sub>2</sub> is to be used. This quantity can be a function of time or constant. This allows the decision maker to only be concerned with one thing, profit, while still having an environmental concern.

Estimating the profitability of the HES with appropriate detail is complex. It is conjectured that the profitability of a HES like the one described above is heavily reliant upon how the system is controlled. One of the HES's key features is being able to take advantage of varying conditions, including the market prices for the inputs and outputs. How the system reacts to these changes is in the control scheme. The performance of the HES may increase for a different control scheme, and still yield a negative overall profit for various reasons. For example, due to the thermal cycling of multiple components, such as the turbines and heat exchangers, the overall life of these components can be drastically reduced in practice. This will be taken into account with additional maintenance and replacement costs. These costs may cause the plant to shut down or operate at reduced capacity factors more often than a stand-alone system. This is why it is important to optimize with respect to profitability as compared to only system performance. The control scheme is not the only alternative for designing optimized HES.

In addition to optimizing the current HES, additional avenues to increase profit will be explored. One area for exploration is that of varying the architecture of the HES for potential increases in profitability. For example, it may be the case that a system that uses a core and a non-core non-renewable load will result in a superior overall profitability for the system. Testing cases such as this may also be considered, however these other architectures will also need to be optimized, making the process difficult. Other methods that can further increase the profitability of the HES will also be explored.

Another alternative is, instead of assuming constant conditions and simulating the current system under those conditions, allowing for the system to

change or expand. This can also increase profit. One method that does this is Real Options Theory. Real Options Theory operates under the premise that evaluating the profitability of a design based on the average operating conditions is flawed [4]. In addition, a system will normally not stay the same throughout its life cycle, as conditions surrounding the system change, the system can be augmented in order to expand or vary how the system acts. This occurs in real life. For example, managers may elect to increase the size of a plant after it has already been built and is in operation. By taking this into account when first building a design, overall costs can be reduced with overall profitability and flexibility being increased.

## 6 Acknowledgments

The work presented in this paper has been supported and funded by Idaho National Laboratory, Energy Security Initiative, under Contract No. 00131206.

## References

- [1] Casella, F., and Leva, A., 2003, "Modelica Open Library for Power Plant Simulation: Design and Experimental Validation," *Proceeding of the 2003 Modelica conference, Linköping, Sweden*.
- [2] Cengel, Y. A., 2007, *Heat and Mass Transfer: A Practical Approach*, McGraw-Hill.
- [3] Chan, H. L., 2000, "A New Battery Model for Use with Battery Energy Storage Systems and Electric Vehicles Power Systems," *Power Engineering Society Winter Meeting, 2000. IEEE*, IEEE, **1**, pp. 470-475.
- [4] de Neufville, R., and Scholtes, S., 2011, *Flexibility in Engineering Design*, The MIT Press, Cambridge, Massachusetts 02142.
- [5] Forsberg, C., 2013, "Nuclear Renewable Futures Employing Nuclear Process Heat," in *INL Hybrid Energy Workshop, Department of Nuclear Science and Engineering*, Massachusetts Institute of Technology.
- [6] Garcia, H. E., Mohanty, A., Lin, W.-C., and Cherry, R. S., 2013, "Dynamic Analysis of Hybrid Energy Systems under Flexible Operation and Variable Renewable Generation—Part I: Dynamic Performance Analysis," *Energy*.
- [7] Garcia, H. E., Mohanty, A., Lin, W.-C., and Cherry, R. S., 2013, "Dynamic Analysis of Hybrid Energy Systems under Flexible Operation and Variable Renewable Generation—Part II: Dynamic Cost Analysis," *Energy*.
- [8] Kim, Y.-H., and Ha, H.-D., 1997, "Design of Interface Circuits with Electrical Battery Models," *Industrial Electronics, IEEE Transactions on*, **44**(1), pp. 81-86.
- [9] Modelica, 2012, "Modelica@ - a Unified Object-Oriented Language for Systems Modeling," in *Language Specification Version 3.3*, <https://www.modelica.org/documents/ModelicaSpec33.pdf>.
- [10] Moran, M. J., and Shapiro, H. N., 2004, *Fundamentals of Engineering Thermodynamics*, John Wiley & Sons, Inc.
- [11] National Renewable Energy Laboratory, U. S., 2006, "Western Wind Resources Dataset."
- [12] National Renewable Energy Laboratory, U. S., 2010, *Homer (the Hybrid Optimization Model for Electric Renewables)*, <http://homerenergy.com/>.
- [13] Petersson, J., Isaksson, P., Tummescheit, H., and Ylikiiskilä, J., 2011, *Modeling and Simulation of a Vertical Wind Power Plant in Dymola/Modelica*, Master's Thesis Thesis, Department of Industrial Electrical Engineering and Automation, Lund University.
- [14] Shaahid, S. M., and El-Amin, I., 2009, "Techno-Economic Evaluation of Off-Grid Hybrid Photovoltaic–Diesel–Battery Power Systems for Rural Electrification in Saudi Arabia—a Way Forward for Sustainable Development," *Renewable and Sustainable Energy Reviews*, **13**(3), pp. 625-633.
- [15] Vitali, D., and Ricci, R., 2013, "Design, Testing and Simulation of Hybrid Wind-Solar Energy Systems."

# Dynamic Modeling of Small Modular Nuclear Reactors using MoDSim

Richard Hale<sup>1</sup>

Sacit Cetiner<sup>1</sup>

David Fugate<sup>1</sup>

Lou Qualls<sup>1</sup>

John Batteh<sup>2</sup>

Michael Tiller<sup>3</sup>

<sup>1</sup>Oak Ridge National Labs, Oak Ridge, TN USA

<sup>2</sup>Modelon, Inc., Ann Arbor, MI USA

<sup>3</sup>Xogeny, Inc., Canton, MI USA

halere1@ornl.gov

cetinerms@ornl.gov

fugatedl@ornl.gov

quallsal@ornl.gov

john.batteh@modelon.com

michael.tiller@xogeny.com

## Abstract

As part of the advanced small modular nuclear reactor (AdvSMR) R&D program, Oak Ridge National Laboratory (ORNL) is developing a Dynamic System Modeling Tool (MoDSim) to facilitate research and development related primarily to instrumentation and controls (I&C) studies of small modular reactors (SMRs).

The primary objective is to produce a demonstration product of a dynamic system modeling tool for SMRs. Functional Mockup Interface (FMI) has been used to allow the development of scoping models for non-Modelica users. This tool includes a web-based interface using Xogeny's FMQ platform for model configuration with local application deployment for simulation using FMI Add-in for Excel from Modelon; this toolchain is designed to allow plug and play access for users of various skill levels. The web-based interface allows true web-based access and solutions without requiring local applications. The initial installation of this tool has been tested on a liquid-metal small modular reactor (ALMR) concept modeled using Dymola and exported via FMI. This tool allows simulation to be performed within Excel without expertise in the native simulation language (Modelica) or model development and simulation environment (Dymola). This toolchain fulfills the Department of energy (DOE) project scope goal of developing a tool "*in a common and familiar environment to support a range of research activities requiring dynamic behavior simulation, modeling tools with easily re-configurable modules that reduce data input to typically available system level plant data*".

*Keywords: nuclear reactors; thermofluid systems; Dymola; Excel; web-based simulations; Advanced Liquid Metal Reactor.*

## 1 Introduction

Small modular nuclear reactors (SMRs) are modular nuclear power plants that are smaller (300 MWe or less) than current-generation base load plants (nominally 1,000 MWe or larger). SMR designs may include factory-fabricated reactors that can be transported by truck or rail to a nuclear power plant site. Small modular reactors offer the advantages of lower initial capital investment, intrinsic safety, scalability, and siting flexibility at locations unable to accommodate larger reactors. In addition, SMRs offer the potential to construct and operate a first reactor and later add more reactor modules at the same site (phased construction). This approach offers economic advantages but could generate numerous configurations of reactor modules, power conversion, and heat sink that will require unique capabilities from control and support systems.

The objectives of this project are to establish a configurable framework for the development of a dynamic simulation environment for SMRs using pre-developed simulation modules, initiate the development of selected modules, and demonstrate their use within an initial integration framework.

The DOE Work Scope includes the development of an interface and workflows (or guided processes) that allow the creation of self-consistent reactor power system designs without in-depth knowledge of the native modeling language or simulation environment. This interface must allow the user to provide input information, process input information

derived from the ruleset (i.e., preconfigured inputs based upon the user's choices) and create executable files to calculate a time-dependent response for the system. Currently the workflows established in the interface for novice users are limited to configured architectures that have been precompiled in Dymola. Novice users can select the subsystems and modify parameters for the generation of results without requiring knowledge of Dymola or Modelica. Additionally, the current interface tool also allows advanced users to create new systems, new models, even new architectures via access to source code in GitHub and use with Dymola. GitHub (<https://github.com/>) is a web-based platform that allows users to share, modify, and comment on code for project collaboration. The use of GitHub integration provides a framework for advanced users to push changes back to the ORNL project team and collaborate as part of the Open SMR community.

The ability to configure models easily (plug-and-play) with software tools requires a formal model architecture. This configurable model architecture is required to enable configuration via web applications. The configurable architecture also enables potential scripting of executable system configurations and FMU generation. The web-based application makes use of this hierarchical architecture structure to simplify the creation of new models from assembled components.

While it is difficult within the page length constraints to include all aspects of the project, it is very important that both unique aspects of this project be described within this paper. Thus, this paper includes the application of this approach to a novel space-advanced nuclear reactors as well as the generation of FMU simulations and application of web-based solutions. Unfortunately, a detailed discussion of both aspects is not possible within the publication constraints. Additional detail will be provided in further publications.

## 2 SMR Dynamic Modeling

### 2.1 LANGUAGE (Modelica)

Traditionally, modeling of complex reactor systems has been based on extensive, complex Fortran-based subroutines. Considerable time and effort were necessary to understand and manipulate these models. In contrast, the Modelica language has built-in features and open-source-toolsets for modeling fluid power systems. Physical systems are modeled by connect-

ing classes of components in series or parallel, and by specifying the important parameters of the objects. Fluids are defined as special classes that model the medium behavior using semi-empirical correlations or first-principles equation-of-state models that yield the full state of the medium as a function of two known states (e.g., pressure and enthalpy). These models can be sophisticated enough to support a wide range of operating regimes for fluids at the cost of computation time—and sometimes numerical stability. These property functions are frequently called during execution of simulations to calculate various engineering variables such as heat transfer coefficients and friction factors. The properties of water are already built into the standard Modelica Media library for use with the Modelica Fluid library. In addition to the Modelica Fluid Library, this project has made extensive use of the ThermoPower Library developed by the University of Milan [1]. The built-in water class has a complex set of routines that support working with water from sub-cooled to supercritical regions. As part of the project, ORNL implemented a number of heat transfer media, including liquid salts (flibe, flinak,  $\text{KFZrF}_4$ ) and liquid metals (sodium, NaK, PbBi eutectic) for use in Modelica. The Modelica models were based on the components and systems presented in [3] for the Power Reactor Innovative Small Module (PRISM) design concept. For validation and verification purposes, the results from running these models were compared with the results documented for the PRISM concept.

### 2.2 SOLVER (Dymola)

The simulation development and execution platform chosen for this SMR modeling effort is Dymola [4]. Dymola is a commercial modeling simulation environment developed by Dassault Systemes. Dymola has been used extensively within the automotive, aerospace, robotics and process system disciplines in part due to the extensive set of libraries developed and available within Dymola. The libraries include thermal and fluid dynamics. As a full simulation environment, a graphical interface to the model objects, as well as tools for the compilation and debugging of the models and analysis of the results are also included. To meet the project objective to provide access to dynamic simulation capability without special tools, export of Functional Mockup Units (FMUs) from Dymola is used in conjunction with FMI Add-in for Excel (FMIE) [5] from Modelon to allow simulation in Microsoft Excel. Co-simulation FMUs are generated for dynamic modeling in FMIE.

## 2.3 Architecture

Modelica and Dymola provide a highly convenient means to create reconfigurable model architectures which include high-level rules for systems, interfaces, and connections. As shown in Fig. 1, the ALMR Power Reactor Inherently Safe Module (PRISM) end-to-end plant systems architecture includes the following systems:

1. Direct Reactor Auxiliary Cooling System (DRACS),
2. Primary Heat Transport System (PHTS),
3. Intermediate Heat Exchanger (IHX),
4. Intermediate Heat Transport System (IHTS),
5. Steam Generator (SG),
6. Power Conversion System (PCS), and
7. Electrical Grid.

These systems will be defined later in the report in more detail. In addition to the plant systems, the following modules are included in the architecture:

1. Event Driver (ED),
2. Control System (CS), and
3. Communication bus.

The ED module generates signals for exciting the system. It performs two basic functions:

1. operational transients and
2. injection of faults.

Operational transients include power changes, load following commands, and intentional reconfiguration. A fault injection function introduces failures or performance degradation in identified plant components. Anticipated operational occurrences (AOOs) are handled through the fault injection function. The CS module performs continuous-time control functions for all actuation interfaces in plant systems based on sensory feedback.

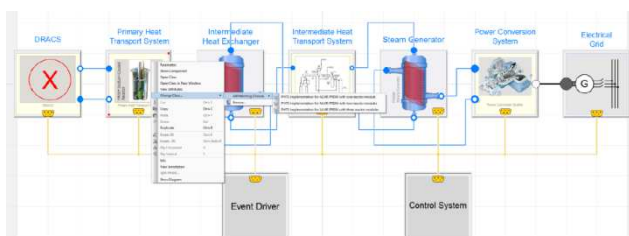


Fig. 1 SMR modeling architecture

## 2.4 Models

A small modular reactor consists of several typical components and subsystems. These include the reac-

tor core, piping and primary pumps that make up the reactor and primary systems. Additional piping, heat exchangers, pumps and steam generator are also included that make up the intermediate loop subsystem. Finally, turbines, and process system components are included in the power conversion system as well as generators and other grid related components in the grid subsystem. A discussion of the specifics of these systems is included in the sections below.

The dynamic models are implemented by equations in Modelica using a hierarchical modeling approach. The component models are often based upon various model libraries that are available as open source or as commercial products. Additionally, a user can create component models from their own Modelica code to supplement the existing libraries. Some of the reactor modeling includes component models created by the ORNL team.

The ALMR reactor design that has been modeled is based on the variation proposed by GE in the mid-1990s and documented in GEF-00793 [3]. The end-to-end system models developed are based on the systems, components, and equations included in this reference. For validation and verification purposes, the results from running these models were compared with the results documented for the PRISM concept. System behavior was confirmed compared to previous model studies.

### 2.4.1 Reactor Core

Most of the component models for SMR's are specific implementations of traditional thermal power components. However, the reactor core is unique and requires special consideration. As a result, the equations for modeling the core will be considered.

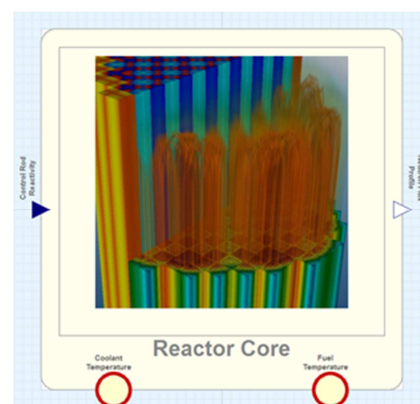


Fig. 2 Reactor core component

The prompt portion of normalized heat generation is implemented with the *point kinetics* equations.

$$\frac{dn}{dt} = \frac{\rho_t - \beta}{\Lambda} n(t) + \frac{1}{\Lambda} \sum_{i=1}^6 \beta_i c_i(t) \quad (1.a)$$

$$\frac{dc_i}{dt} = \lambda_i [n(t) - c_i(t)] \quad (1.b)$$

where  $n(t)$  is normalized reactor power,  $c_i(t)$  is the normalized concentration of the  $i$ -th group delayed neutron precursor,  $\beta_i$  is the fraction of the  $i$ -th group precursor,  $\lambda_i$  is the decay constant for the  $i$ -th group precursor,  $\rho_t(t)$  is the total reactivity, and  $\Lambda$  is mean neutron generation time. The rate equations are subject to steady state initial conditions, i.e.,  $\dot{n} = \dot{c}_i = 0$ .

The delayed portion of normalized heat generation is implemented with the following:

$$Q_{n-decay} = 0.1 \left[ (t + 10)^{-\frac{1}{5}} - (t + T_s)^{-\frac{1}{5}} + 0.87(t + T_s) + 2 \times 10^7)^{-\frac{1}{5}} - 0.87(t + 2 \times 10^7)^{-\frac{1}{5}} \right] \quad (2)$$

where  $t$  is time after shutdown and  $T_s$  is the operation time prior to shutdown—both in seconds.

The reactivity feedbacks are modeled as follows:

$$\rho_f = \alpha_f (T_{fe} - T_{f0}) \quad (2.a)$$

$$\rho_c = \alpha_c (T_{ce} - T_{c0}) \quad (2.b)$$

$$\rho_t = r\theta_{CR} + \rho_f + r\theta_c \quad (2.c)$$

where  $\rho_f$  is the fuel Doppler reactivity feedback,  $\rho_c$  is the coolant density reactivity feedback, and  $\rho_t$  is the total reactivity feedback.

The axial neutron flux shape is implemented as follows:

$$\Phi(z) = \Phi_{max} \cos\left(\pi \frac{z}{L}\right) \quad (3)$$

where  $z$  is the axial spatial variable that varies between  $-L/2$  and  $L/2$ , and  $L$  is the active length of

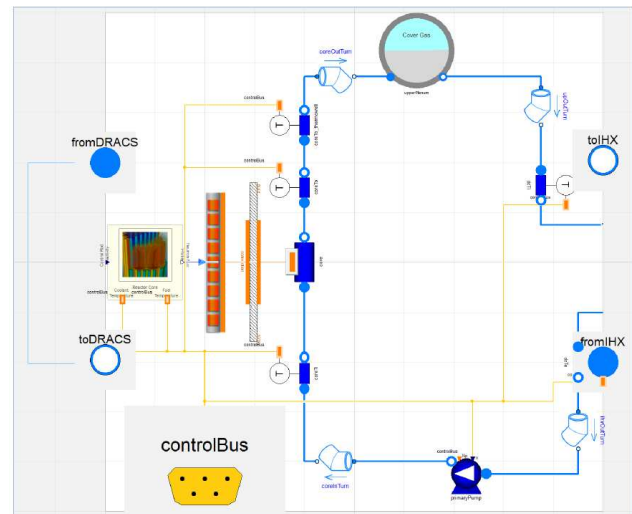
the fuel element. This particular representation is good for a wide range of reactor classes, but not quite so for Boiling Water Reactors, where flux shape is significantly skewed due to the boiling process.

For a given number of axial nodes, the integrated nodal flux is calculated with the following expression:

$$\begin{aligned} \phi_i &= \int_{-\frac{L}{2} + i\delta}^{-\frac{L}{2} + (i-1)\delta} \Phi_{max} \cos\left(\pi \frac{z}{L}\right) dz \quad (4) \end{aligned}$$

## 2.4.2 Primary Heat Transport System

The PHTS includes a reactor core model, a fuel pin thermal model, a discretized one-dimensional core flow model, a cover gas model, and a mechanical pump model. The Dymola/Modelica model diagram layer is shown in Fig. 3. The performance parameters of the mechanical pump are taken from the ALMR PRISM primary EM pump performance characteristics curve at 650 V, which is the operating point of all EM pumps under normal conditions. The mechanical pump will later be replaced with an EM pump model in the subsequent phases of the project. The model also includes four 90-degree turn elements to incorporate irrecoverable pressure losses due to turns and sudden expansion and contractions in the flow path. These loss elements are intended to match the total pressure drop during normal operating conditions across the core to the ALMR PRISM specifications.



**Fig. 3 SMR Primary Heat Transport System**



### 2.4.3 Intermediate Heat Transport System

The ALMR PRISM baseline design includes two IHXs connected to a single IHTS. The IHTS flow loop (Figure 4) also includes a mechanical sodium pump, a sodium expansion tank, a sodium-to-water steam generator, and a sodium dump tank. The sodium dump tank is not modeled in the IHTS implementations. However, this tank should not have an impact on simulation results as this safety system is activated only under accident conditions where the sodium in the IHTS reacts with steam or air.

The ALMR PRISM intermediate sodium pump is a vertically oriented, single-stage, double-suction, free-surface, centrifugal pump driven by a constant-speed, 4000-hp (approximately 3000 kW), air-cooled induction motor. An auxiliary pony motor drive provides low flow (10%) capability for decay heat removal and other low-power or standby conditions. Its baseline design is to provide a flow of approximately 1900 kg/s at 96-m static head and 282°C. The main drive motor generates 3000-kW power at 1750 rpm.

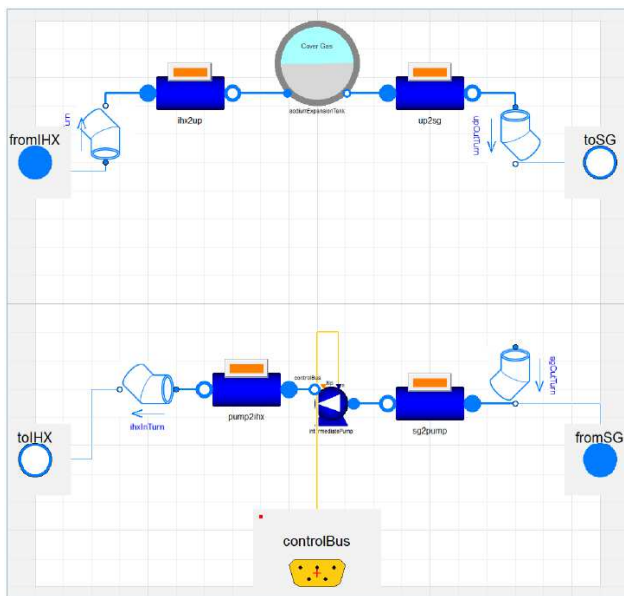


Fig. 4 SMR Intermediate Heat Transport System

### 2.4.4 Intermediate Heat Exchanger

The ALMR PRISM Intermediate Heat Exchanger (IHX) is modeled using two one-dimensional flow elements connected through a radial metal tube element, as shown in Fig. 5. The metal tube element is radially and axially discretized; the number of nodes can be changed by the user. The material properties, that is, density, thermal conductivity, and specific

heat capacity, can also be defined and changed using the Dymola dialog interface.

The default medium on both sides of the IHX is sodium. However, both fluid media can be selected via the system dialog interface. A number of fluid models are provided in the drop-down menu for both shell-side fluid and tube-side fluid, such as incompressible sodium (default), compressible sodium, incompressible NaK, and compressible NaK. Additionally, the menu selection includes a number of incompressible salt models; however, selecting incompatible media may result in numerically unstable configurations as the default initial and boundary conditions are specified for the ALMR PRISM design in the implementations. In the later phases of the project, the menu selections will be bound to a certain reactor class logic, which will limit the number of fluid media options for a certain implementation. The fluid flow and the heat transfer models in the shell side and the tube side can also be selected via the system dialog interface.

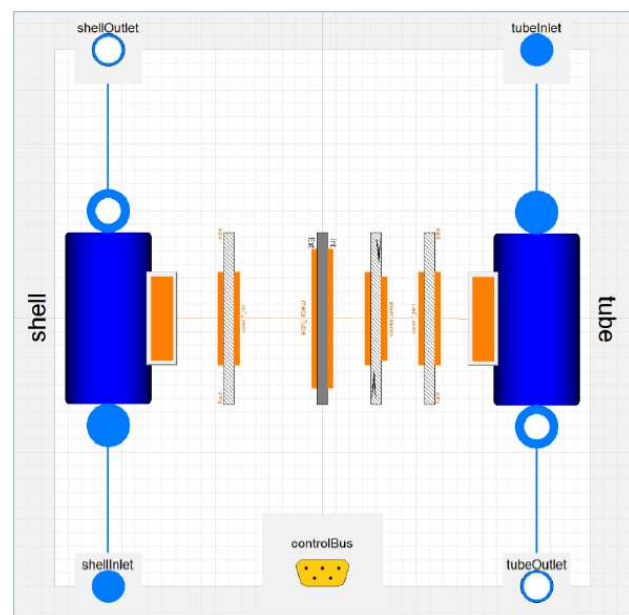


Fig. 5 SMR Intermediate Heat Exchanger

### 2.4.5 Steam Generator

The ALMR PRISM steam generator is a vertically oriented, shell-and-tube counter-flow heat exchanger with water/steam on the tube side and sodium on the shell side. The tubes are straight and of double-wall construction. The double-wall tubes provide improved reliability by significantly reducing the probability of water or steam leaking into the sodium.

The Modelica implementation of the ALMR PRISM steam generator is shown in Fig. 6. The ALMR PRISM steam generator is not a helical coil construc-

tion; therefore, it uses different analytical formulations for representation of fluid heat transfer and pressure drops. Furthermore, in contrast to the IHX model as shown in Fig. 5, the steam generator model employs two cylindrical tubes, one inner tube one outer tube, to account for the double-walled construction of the tube bundle. The other tube model of the steam generator implementation introduces a small thermal resistance to account for small interface separation.

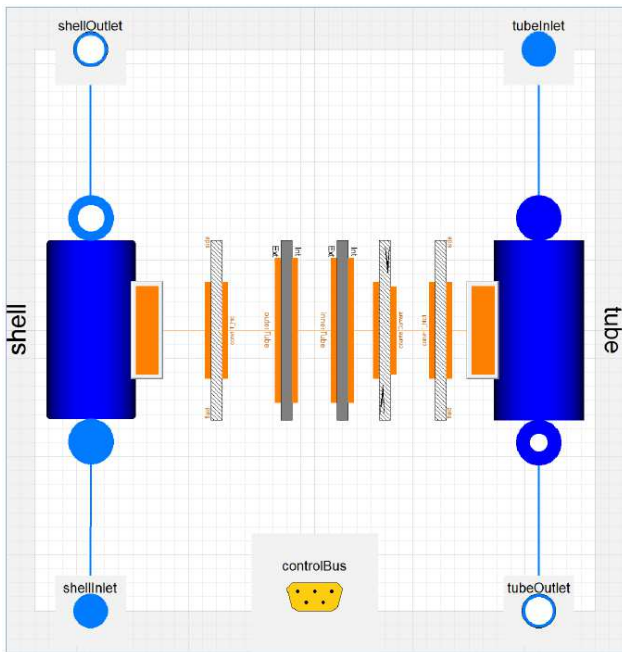


Fig. 6 SMR Steam Generator

### 2.4.6 Power Conversion System

The power conversion system model (Fig. 7) is comprised of several components, including the turbines, generator, flow control valve, and pump. Standardized modules that include representative equations/curves are used for these modules. No specialized modules for the ALMR were developed. Current implementation is a simplified version of the actual power conversion system where the feedwater heater trains, blowdown coolers, and the deaerator are eliminated. The implementation also uses a constant rotor input (on the left side of the rotor shaft as a boundary condition represented by the “shaftSpeed” element) to eliminate additional control requirements to maintain shaft angular speed. The rotor mass and rotational mechanics are represented by the rotor element.

The “steamInlet” and “condensateReturn” ports are implemented as vector objects whose number of elements must match the number of steam

generators, which in turn matches the number of reactor modules in a power block. In the current implementation, up to three steam lines can be allocated; however, this limitation will be removed later. Each steam line contains a “stopValve” element (shown as “stopVx”) to control steam flow from each stream. It should be noted that the “stopValve” elements are not modulating elements; they have binary states of OPEN or CLOSED. The valves’ opening and closing dynamics are represented by a linear ramp with a user-specified time constant to switch from one state to another. The streams from three steam generators are mixed via two “flowJoin” elements.

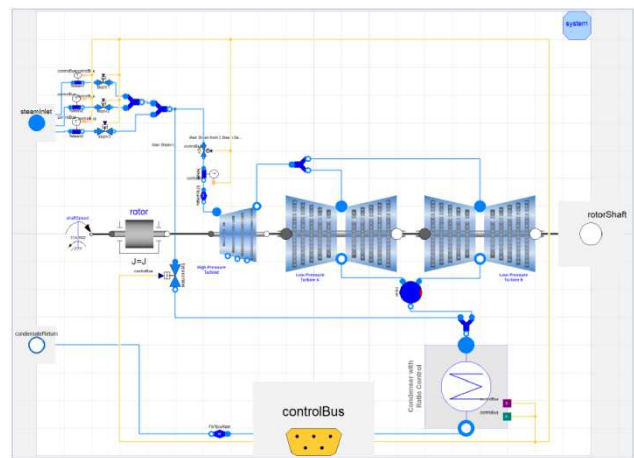


Fig. 7 SMR Power Conversion System

### 2.4.7 Grid

A simplified grid model has been developed. The Modelica implementation of the generator and the electrical grid is shown in Fig. 8. In the current implementation, the electrical generator is modeled as an ideal synchronous generator where the frequency in the electrical connector is the electromotive force (emf) of the generator. The frequency of generated electrical current is by default 60 Hz but can also be adjusted by the user. It should be noted that the generator frequency should match the grid frequency; otherwise, it will cause a generator trip and disconnection from the grid, which will in turn initiate a turbine trip.

The electrical grid is modeled with the *swing equation*, which calculates the *load angle* as a function of differences between the mechanical torque (provided by the turbine shaft) and the electrical torque (resistant torque generated by the electrical load). The load angle dictates the amount of power that can be transferred to the grid. The power sensor and the frequency sensor are used to monitor the load

angle of the network. If the frequency gets out of range, an anticipatory automatic trip is initiated for the turbine and the generator.

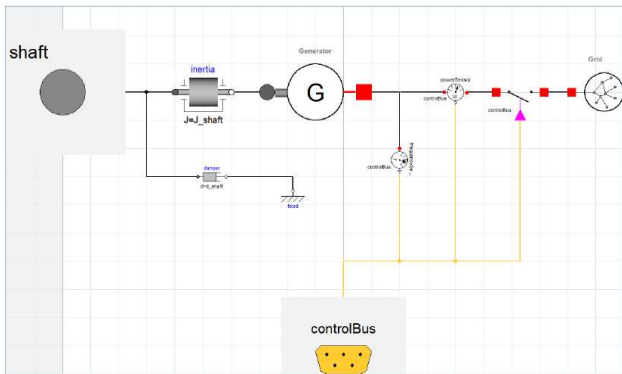


Fig. 8 SMR Grid System model

2.4.8 I&C

The purpose of this project is to design a system of modeling tools that allows rapid assessment of control system strategies overlaid upon reactor system models. For demonstration purposes, two different control strategies were chosen to allow multiple selection options to exist in the modeling tool: Strategy #1 – temperature regulation and Strategy #2 – temperature difference regulation.

The purpose of developing these systems models is to investigate potential control strategies for advanced SMR concepts. The I&C studies and concept development include identification of the model global outputs, determination of measurements, control actuation, and control methods and algorithms for a given system. The dynamic model of the process and plant physics provides model results and outputs for candidate measurements (system observability) and model inputs for candidate actuation (system controllability). The dynamic model also provides insights into the system time constants, degree of inherit stability, degree of coupled behavior, and input-output sensitivity. Figures 9-11 show the sensing instruments, model architecture and control logic associated with this simplified I&C control module that is overlain on the reactor systems model.

Figure 12 illustrates an example simulation in Dymola of a control system during energy demand changes. The PHTS primary pump and IHTS intermediate pump and record control rod reactivity are all actuated to regulate the reactor system temperatures as the heat sink changes. The generation of these outputs using the systems modeling approach with Modelica is relatively straightforward for the practiced simulation engineer. The goal of this project is to extend this capability to those engineers

who are not as familiar with these modeling and simulation environments.

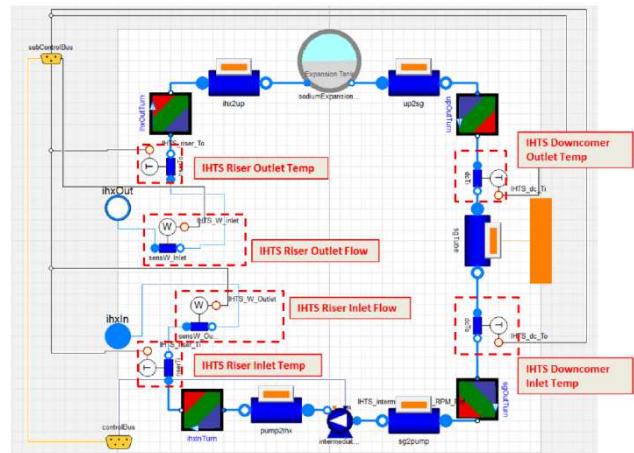


Fig. 9 SMR I&C sensing instruments

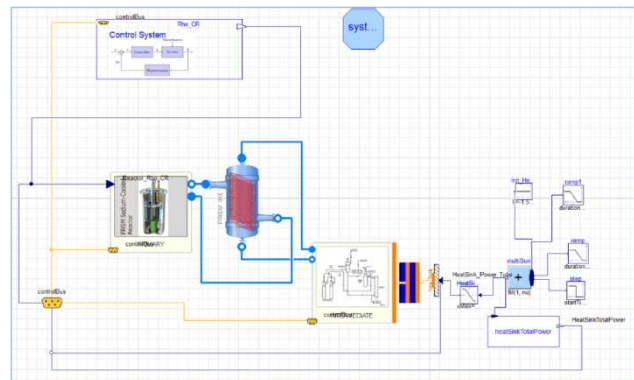


Fig. 10 SMR I&C

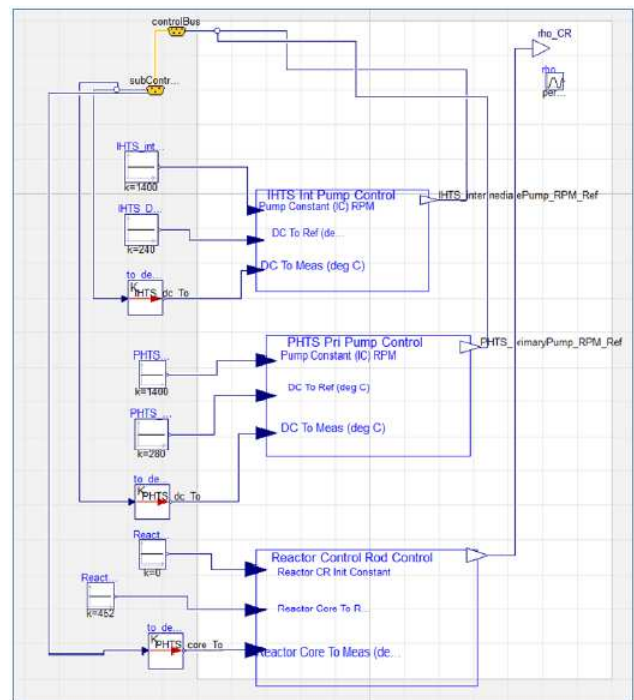


Fig. 11 SMR I&C architecture

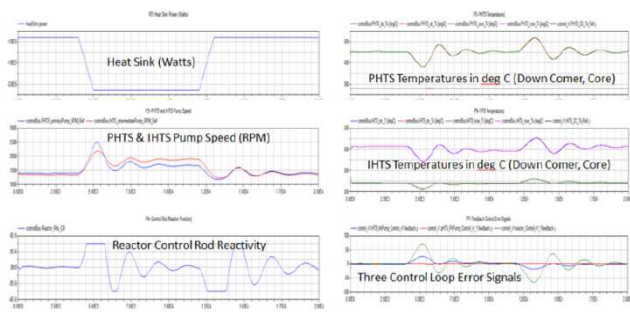


Fig. 12 Initial control system testing results

### 2.5 FMI Add-in for Excel

Although simulation is a valuable tool for science and engineering, few scientists and engineers have the proper skill set or day to day need for modeling and simulation. As a result, the investment of time and resources to establish and maintain a modeling and simulation capability is often not justified. An open framework would provide an alternative approach which will open modeling and simulation capabilities to activities that cannot justify the investment in traditional modeling tools.

The development of these models initially requires some knowledge of Modelica and Dymola. Although most engineers may not be familiar with Modelica or Dymola, many engineers are familiar with and fairly skilled in Excel. Therefore, the project has built the initial interface tool around an Excel platform that allows the simple and intuitive generation, manipulation, plotting, and reporting of results using the FMIE [5] tool developed by Modelon. This tool (Fig. 13) allows the integration of Excel with compiled co-simulation FMUs for simulation. Many simulation environments (not just Modelica based) now allow the export of these FMUs. The use of the open standard FMI provides an opportunity for a flexible toolchain in terms of tools used for core model development, simulation, and engineering analysis along with associated licensing.

FMIE can process a model that has been compiled into an FMU by simulation platform like Dymola in a manner that facilitates performing dynamic simulations in Excel. The tool also supports changing model parameters, adjustment of the simulation start and stop times, for single or multiple simulations which can be run in parallel. The resultant input and output data variables can be selected for display in graphical or tabular form in a worksheet. FMIE also provides a scripting API which can be used for automation of analysis set-up, simulation, plotting, and post-processing.

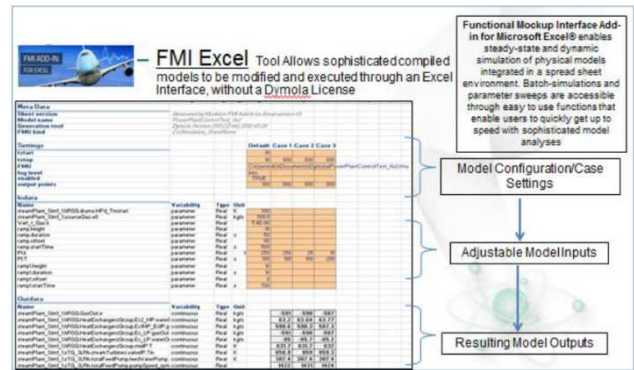


Fig. 13 Modelon FMIE interface

### 2.6 Collaboration

The initial user interface for MoDSim has been written in Excel (Fig. 14). This makes for easy integration with FMIE used to simulate the Modelica models outside of Dymola. Excel macro options that allow for the modification of inputs, the display of outputs, and the generation and plotting of results (Fig. 15) are provided along with the ability to display the metadata about the simulation. This interface tool, however, only serves as the prototype in terms of features and layout for the web application interface that is being developed. The principal limitation associated with the prototype Excel interface is its use as a local application on a machine. The widespread collaboration that is the intended goal is only truly achievable from a web-based platform.

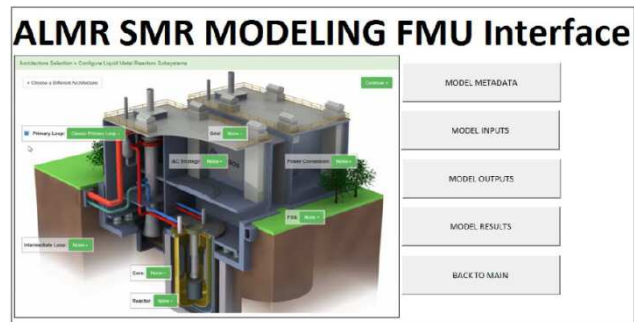


Fig. 14 SMR Excel interface

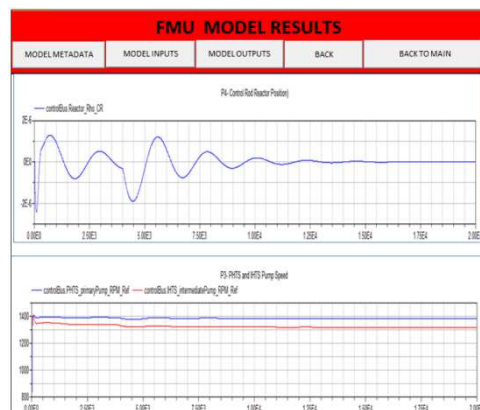


Fig. 15 SMR Excel interface - results

The heart of the collaborative effort is based on using an online open-source version control, issue tracking, and file sharing application. The application chosen is GitHub. Users can obtain a GitHub account and request access to the ORNL SMR folder (ORNL-WebSMR) displayed in Fig. 16. Access to this folder allows the user to work directly in the text-based Modelica models and make modifications, comments, etc. on the models. Additionally, issues can be raised and tracked as well as versions of the models created, archived, and documented. This allows collaboration within the user community.

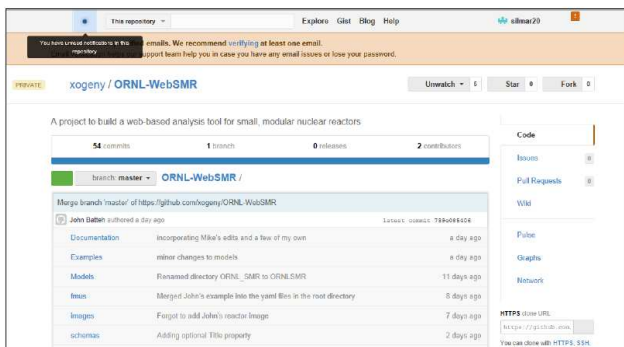


Fig. 16 SMR GitHub model repository

The start page of the SMR web application built by Xogeny, Inc. is shown in Fig. 17. The web platform utilizes the architecture described in Section 2.3 along with configuration files that provide architecture, subsystem choice, parameters, and output information for use in generating the actual web application. These files are generated in the YAML (Yet Another Markup Language) language. The user can browse the available reactor architectures and then selects the one with which to work. Following the selection of the reactor architecture, the user can then configure the subsystems in the architecture as shown in Fig. 18. After subsystem selection, the user then has the option to modify the parameters in each subsystem (web app is populated with the default parameters from the FMU) as shown in Fig. 19.

The current workflow is then to generate a simulation with FMIE as shown in Fig. 20. The web application provides a download of a customized Excel sheet developed by Modelon that utilizes the scripting API in FMIE to automatically download the FMU populated with parameters entered by the user, execute the simulation, and plot the results. The resulting experiment sheet in FMIE can then be used to conduct batch simulations, modify additional parameter values, and automatically compare results between simulations. Figure 21 shows a sample experiment sheet with batch simulation compare. This automation with FMIE provides a one-click work-

flow that couples the web-based configuration and parameter setting with a local simulation running in Microsoft Excel.

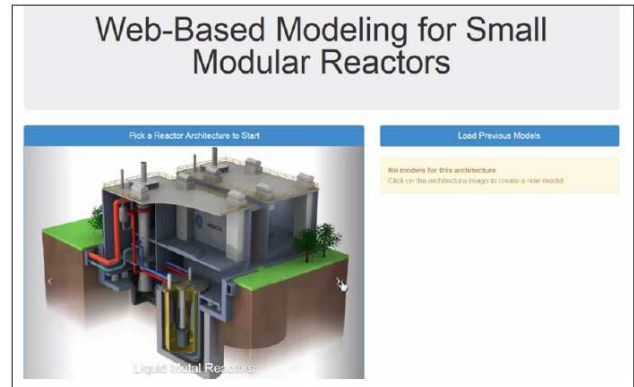


Fig. 17 SMR web app start page

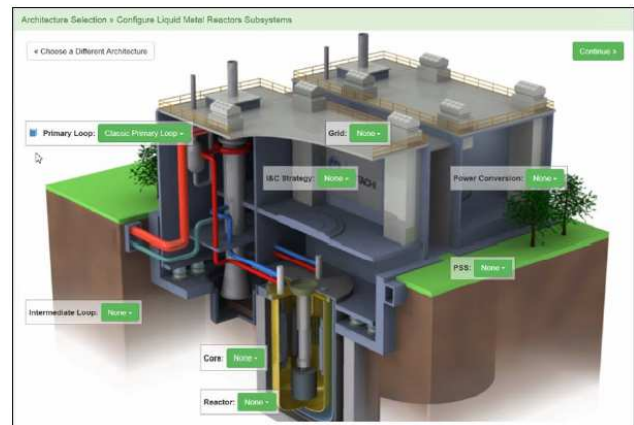


Fig. 18 SMR model configuration

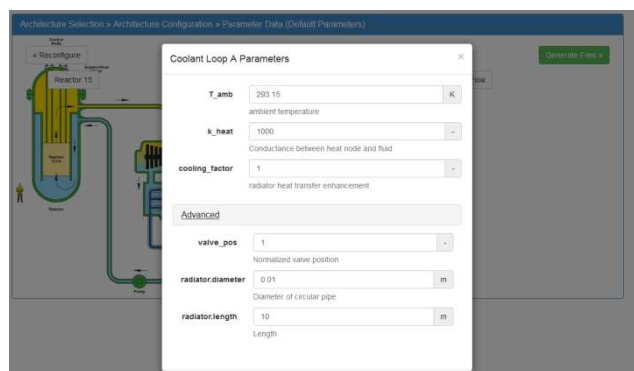


Fig. 19 Parameter specification

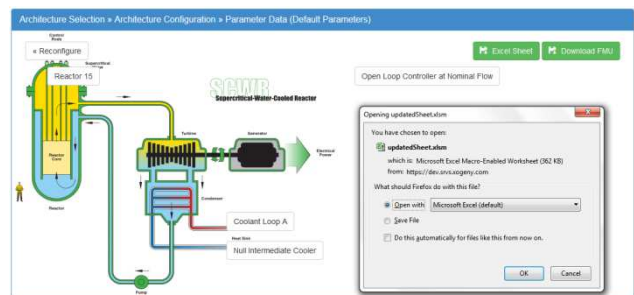


Fig. 20 Simulation generation with FMIE

Model		Generated by Modelon FMI Add-in for Excel version 1.2										
Sheet version		ORNL Demo Systems_FvHeatMassFlowOpen										
Model name		Dymola Version 2013 FD01 (64-bit), 2012-10-18										
Generation tool		C:\Users\jbatteh\AppData\Local\Temp\FMUD12C\mp_K35BTU3\HEM7.Fmu										
FMI kind		CoSimulation_Standard										
Number of processes		0										
Settings		Default	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10
Start time		0										
Stop time		1000										
FMIU		Info										
Log level		Info										
Enable		TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Output points		100										
Timeout		0										
Inidata		Name	Variability	Type	Unit							
		reactor.rod.T	continuous	Real	K	293.15		300				
		reactor.Q_reactor	parameter	Real	W	1500		1750				
		reactor.C_rad	parameter	Real	1/K	10						
		coolant_loop.radiator.length	parameter	Real	m	10						
		coolant_loop.radiator.diameter	parameter	Real	m	0.01						
		coolant_loop.h_heat	parameter	Real	1	1000						
		coolant_loop.wiring.gas	parameter	Real	K	293.15						
		coolant_loop.T_amb	parameter	Real	K	293.15						
		coolant_loop.cooling_factor	parameter	Real		1						
		controller.mflow_pump	parameter	Real	kg/s	0.01						
Outdata		Name	Variability	Type	Unit							
		reactor.node.Q_flow	continuous	Real	W	1499.94		1749.93				
		reactor.rod.T	continuous	Real	K	486.632		519.181				
		coolant_loop.tank.level	continuous	Real	m	1.0098		1.00455				
		coolant_loop.coolant_bus.T_rad_in	continuous	Real	K	326.643		364.188				
		coolant_loop.coolant_bus.T_rad_out	continuous	Real	K	307.842		310.412				
		coolant_loop.coolant_bus.mflow_pump_sensed	continuous	Real	kg/s	0.01		0.01				
		coolant_loop.coolant_bus.mflow_pump	continuous	Real	kg/s	0.01		0.01				
		controller.controlBus.coolant_bus.mflow_pump	continuous	Real	kg/s	0.01		0.01				
Message												
			OK	OK	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled

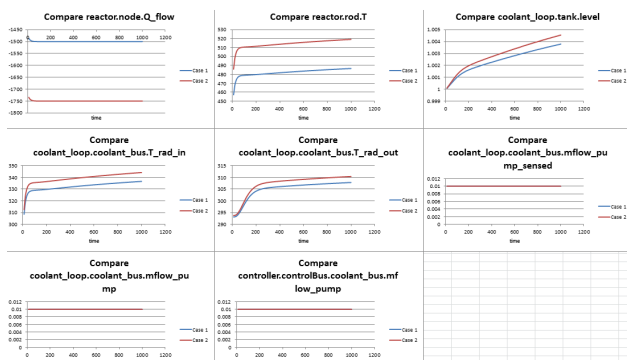


Fig. 21 Sample experiment sheet in FMIE with batch simulation compare

Currently the web platform is an alpha product which still requires local installation of tools, and the alpha version is being used to define the requirements for the beta product. The beta version will be a completely web-based application that can be run from any user's desktop without the need to install any software. ORNL is working with Modelon and Xogeny to facilitate a web-based Modelica simulation application towards this end. The beta product is expected to be out in FY 2015. The goal of the web platform is to support both local and cloud-based simulations within licensing constraints of the development environments. As with any compiled simulation environment, the practical challenges of pre-configured models, initialization, and simulation performance must be handled in a robust way to ensure successful deployment and effective use of the simulation platform.

### 3 Conclusions

A growing number of programs within the DOE portfolio will benefit from the streamlined approach to collaborative modeling that this tool affords. Other actively funded activities in the DOE Advanced Reactors Program are planning to incorporate this tool

into their activities. This tool is well suited for work in the study of hybrid energy systems. ORNL is actively engaged in developing collaborations in the hybrid energy system space that utilized this approach. In particular, ORNL is working with the Idaho National Laboratory to create these hybrid energy system models that can be used to provide high level evaluations of potential energy concepts that make use of multiple energy generation paths. In the next phase of development, we will actively engage these projects and seek additional projects that will also benefit from this tool.

As a final note, the long-term success of this activity will be judged by three items: quality of the models developed, the number of users, and the ease of use. To this end, further development in the web-based platform is an important next step. The web-based platform provides opportunities for cloud-based modeling for both nonsensitive material as well as controlled access for sensitive, proprietary material. Additionally, the ability to create scripts that will allow for rapid assessment of parametric sweeps will simplify the input and output necessary for the user. However, it is critical that multiple projects and programs begin using this tool to generate a library of models that will form the backbone of this collaborative network for SMR dynamic modeling. By generating this simplified modeling platform it is hoped that a new paradigm for systems modeling will be created that will help facilitate the development of the next generation Small Modular Nuclear Reactor, but other complex engineering systems as well.

### References

- [1] F. Casella, A. Leva, "Modelica open library for power plant simulation: design and experimental validation", *Proceedings of the 2003 Modelica Conference*, Linköping, Sweden, November 2003, pp. 41-50.
- [2] ORNL/TM-2013/426, "SMR Dynamic System Modeling Tool Update: September 2013", September 2013.
- [3] GEF-00793, "PRISM Preliminary Safety Information Document", UC-87Ta, December 1987.
- [4] Dassault Systemes, "Dymola 2013 FD01", 2012.
- [5] Modelon, "FMI Add-in for Excel", Version 1.2.1, 2013. <http://www.modelon.com/products/fmi-add-in-for-excel/>

# Modified Multiple Shooting Combined with Collocation Method in JModelica.org with Symbolic Calculations

Evgeny Lazutkin, Abebe Geletu, Siegbert Hopfgarten, Pu Li  
Simulation and Optimal Processes Group, Institute for Automation and Systems Engineering,  
Technische Universität Ilmenau, P.O.Box 10 05 65, 98684 Ilmenau, Germany.  
(evgeny.lazutkin, abebe.geletu, siegbert.hopfgarten, pu.li)@tu-ilmenau.de

## Abstract

This paper presents an efficient and a novel implementation of a combined multiple shooting and collocation (CMSC) algorithm for the solution of nonlinear optimal control problems. The implemented algorithm is a modification of the approach proposed in [17, 18]. The new implementation is done under the JModelica.org framework along with CasADi and Ipopt. The framework uses a symbolic pre-calculation of functions and derivatives. Besides the integration of various components of JModelica.org, Ipopt, and CasADi, the implementation facilitates simpler modeling of optimal control problems along with a choice of options for various linear algebra algorithms. The paper gives a description of the algorithm and elaborates the components of the framework. Numerical experimentations show that the new implementation is efficient in comparison with the published results of other authors.

*Keywords: Nonlinear Optimal Control; Symbolic Automatic Differentiation; Nonlinear Programming; Multiple Shooting; Collocation*

## 1 Introduction

Optimization methods nowadays play pivotal roles in engineering and industrial applications. Most engineering applications are dynamic by nature. Frequently, such dynamic processes have model equations involving large-scale nonlinear differential equations. Hence, the solution of large-scale optimal control problems is difficult to achieve by solving the equations of the optimality conditions. Therefore, the modern approach follows the "first discretize, then optimize" strategy. In this way, the optimal control problem will be transformed into a nonlinear programming problem (NLP). This facilitates the implementation of efficient and state-of-the-art NLP solvers to determine highly accurate approximate optimal solutions to the

continuous optimal control problem.

The direct discretization of optimal control problems through the multiple shooting method was first proposed in [9]. On the other hand, the direct discretization of optimal control problems through collocation methods has been widely used for state constrained optimal control problems (e.g., [7, 10, 13]). The multiple shooting discretization results in block-structured matrices and facilitates easy parallelization of computations. However, the accuracy of the multiple shooting method can be highly augmented if it is combined with the collocation method. Therefore, recently, discretization of optimal control problems through a CMSC method is found to have enormous potentials for the solution of complex large-scale optimal control problems [17, 18].

In order to facilitate the industrial application of complex optimization algorithms, model-based optimization of dynamic systems is recently gaining greater momentum [1, 12]. The aim of this work is to implement the CMSC algorithm in the JModelica.org framework. We first divide the time horizon  $[t_0, t_f]$  into subintervals (finite elements). Subsequently, we use multiple shooting and collocation methods to discretize the optimal control problem and to transform it into an NLP. In our implementation, we use pre-calculated derivatives, i.e., Jacobian matrix in symbolic form by means of CasADi [3, 4]. The nonlinear optimization problem is solved by using Ipopt (**I**nterior **p**oint **o**ptimization solver, [19]). The total optimization tool-chain is provided in the JModelica.org framework [2].

The rest of the paper is organized as follows. Section 2 provides a general form of the considered optimal control problem. Section 3 presents the combined multiple shooting and collocation algorithm. Section 4 describes the implementation of the algorithm under the JModelica.org framework coupled with CasADi and Ipopt and section 5 presents case studies. The

comparative analysis in section 6 shows the high efficiency and viability of our implementation as compared to other implementations. The paper concludes in section 7 with a summary and future research work.

## 2 Problem definition

We consider the following general form of a nonlinear optimal control problem (NOCP):

$$\min_{u(t)} \left\{ J = E(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t), t) dt \right\} \quad (1)$$

$$\text{subject to: } \dot{x}(t) = f(x(t), u(t), t), \quad x(t_0) = x_0, \quad (2)$$

$$g(x(t), u(t), t) = 0, \quad (3)$$

$$x_{min} \leq x(t) \leq x_{max}, \quad (4)$$

$$u_{min} \leq u(t) \leq u_{max}, \quad (5)$$

$$t_0 \leq t \leq t_f, \quad (6)$$

where  $x(t)^\top = (x_1(t), \dots, x_n(t))$  and  $u(t)^\top = (u_1(t), \dots, u_m(t))$  are the state and control vectors, respectively. The functions  $f$ ,  $h$ , and  $g$  are conveniently defined in appropriate function spaces. The dynamics of the process is described by (2) and we have algebraic equality constraints (3). In practical engineering or industrial application, state and control variables are usually bounded. Hence, (4) imposes lower and upper bounds on the states, while (5) defines bounds on the controls. The optimization variables (typically the controls) and state variables must satisfy the model equations, the state and control constraints and the boundary conditions. Furthermore, the optimization task is limited to a time horizon with an initial time  $t_0$  and a fixed final time  $t_f$ . In this paper, time-optimal control problems will not be considered.

The optimal control problem (1) - (6) is an infinite-dimensional problem. Since real-world applications have very complicated structure including nonlinearity and high dimensionality, indirect methods, like methods based on the Pontryagin's principle, are not suitable. Therefore, here the NOCP will be directly discretized by using a combined multiple shooting and collocation (CMSC) method and thereby it will be transformed to a finite-dimensional optimization problem.

## 3 An improved multiple shooting with collocation framework

For the multiple shooting and collocation discretization scheme, first the time interval  $[t_0, t_f]$  is divided into appropriate shooting intervals  $[t_i, t_{i+1}]$ ,  $i =$

$0, \dots, N-1$ . Then, on each shooting interval  $[t_i, t_{i+1}]$ , collocation nodes  $t_i = t_{i0} < t_{i1} < \dots, t_{iN_c} = t_{i+1}$  are defined, so that each state variable  $x_k(t)$  is approximated by the polynomial

$$\hat{x}_k(t) = \sum_{j=0}^{N_c} x_{ij}^{(k)} l_{ij}(t), \quad (7)$$

where  $l_{ij}(t)$  are the Lagrange polynomials

$$l_{ij}(t) = \prod_{\substack{s=0 \\ s \neq j}}^{N_c} \left[ \frac{t - t_{is}}{t_{ij} - t_{is}} \right], \quad j = 1, \dots, N_c, i = 1, \dots, N, \quad (8)$$

and the variables  $x_{ij}^{(k)}$ ,  $j = 1, \dots, N_c$ , represent the state values corresponding to the state variable  $x_k(t)$ ,  $k = 1, \dots, n$ , at the collocation points on  $[t_i, t_{i+1}]$  with the property that  $x_k(t_{ij}) = \hat{x}_k(t_{ij}) = x_{ij}^{(k)}$ . Hence, the combined multiple shooting and collocation scheme transforms the NOCP into an NLP with the additional constraints

$$h_{i+1}^k = \hat{x}_k(t_{(i+1)0}), \quad i = 0, \dots, N-1; k = 1, \dots, n \quad (9)$$

to be imposed along with the discretized constraints (2)-(5). The equations (9) guarantees the continuity of the state trajectories at the end point of each shooting interval. In the following  $h_{i+1}^x = (h_{i+1}^1, \dots, h_{i+1}^n)^\top$  and  $\hat{x}_{i+1}(h_i^x, v_i, t_{i+1}) = (\hat{x}_1(t_{(i+1)0}), \dots, \hat{x}_n(t_{(i+1)0}))^\top$  are used for the sake of brevity. The expression  $\hat{x}_{i+1}(h_i^x, v_i, t_{i+1})$  indicates that the state variables are dependent on the initial state  $h_i^x$ , the controls  $v_i$  and the end time point  $t_{(i+1)0} = t_{i+1}$  on the interval  $[t_i, t_{i+1}]$ . The resulting nonlinear optimization problem can be written as

$$\min_{h_0^x, \dots, h_N^x, v_0, \dots, v_{N-1}} \left\{ E(h_N^x) + \sum_{i=0}^{N-1} L(h_i^x, x_i, v_i) \right\} \quad (10)$$

$$\text{subject to: } h_{i+1}^x - \hat{x}_{i+1}(h_i^x, v_i, t_{i+1}) = 0, \quad i = 0, \dots, N-1, \quad (11)$$

$$G(h_i^x, x_i, v_i) = 0, \quad i = 0, \dots, N-1, \quad (12)$$

$$h_0^x - x_0 = 0, \quad (13)$$

$$x_{min} \leq h_i^x \leq x_{max}, \quad (14)$$

$$u_{min} \leq v_i \leq u_{max}, \quad (15)$$

In problem (10) - (15) on the  $i$ -th shooting subinterval,  $h_i^x$  represents parameterized initial conditions for the vector of state variables,  $\hat{x}_i$  is the state on the collocation point at the end of interval  $i$ , the vector  $x_i$  consists of all coefficients of the collocation polynomials corresponding to the states on the  $i$ -th interval and  $v_i$  are



parameterized control variables. The discretization of the states uses the Radau collocation method. Using the function  $G$ , equation (12) represents the discretized form of the equations (2, 3). The control variables are usually parameterized as piecewise constant functions. E.g., in each subinterval  $[t_i, t_{i+1}]$ ,  $i = 0, \dots, N - 1$ , the control variables are assumed to take constant values and the state trajectories will be approximated by the collocation method. The unique feature of CMSC lies in the fact that it utilizes the advantages of both multiple shooting and collocation methods. Detailed discussions on multiple shooting and collocation methods are found in [8, 9, 16].

In the objective (10) the variable  $x$  does not appear exclusively as an optimization variable. Hence, after appropriate aggregation of variables, at each step of the optimization algorithm, we can solve for  $x$  in terms of  $(h, v)$ , so that we have  $x(h, v)$ . In this way the equality constraints (12) can be eliminated by reducing the number of optimization variables. However, in this work, constraints on the coefficients of the collocation polynomials are not considered inside the collocation intervals, but only at the end-points of the intervals.

Therefore, the problem (10) - (15) can be equivalently written as

$$\min_{h,v} F(h, x(h, v), v) \tag{16}$$

$$\text{subject to : } H(h, v) = 0 \tag{17}$$

$$x_{min} \leq h \leq x_{max} \tag{18}$$

$$u_{min} \leq v \leq u_{max}, \tag{19}$$

where  $H$  in equation (17) provides a compact representation of the equations (11) - (13). Consequently, to solve the nonlinear optimization problem (16) - (19) we use the state-of-the-art optimization solver Ipopt [19]. At each iteration of Ipopt, the nonlinear model equations are solved by using a local Newton algorithm along with a linear algebra solver for the determination of the Newton-steps to determine state variables for given values of  $h$  and  $v$ . Future investigations will consider the implementation of a globalized Newton method with appropriate linear algebra solvers.

All function values and gradients, required in the optimization framework, are pre-calculated and made available in symbolic form. Symbolic derivatives are calculated by using CasADi and stored in matrices or vectors to facilitate faster accessibility. For this, it is needed to compute the sensitivities  $\frac{\partial F}{\partial v}$ ,  $\frac{\partial F}{\partial h}$ ,  $\frac{\partial H}{\partial v}$ , and  $\frac{\partial H}{\partial h}$  in symbolic representations. Further details are found in [17, 18].

The computational framework is summarized

graphically in Fig. 1 and Algorithm 1 provides a description in general terms. More detailed discussions on the CMSC method are found in [17, 18].

**Algorithm 1** : A general CMSC

- 1: **Input**: Time horizon, number of subintervals, number of state and control variables, lower and upper bounds for the control and state variables, model equations, objective function, initial guesses, optimizer options.
- 2: Initialization of the NOCP
- 3: Define continuity and path constraints
- 4: Initialize the n-point collocation for each subinterval
- 5: Compute states at collocation points and sensitivities
- 6: Compute objective function and gradient
- 7: Compute constraint function values and Jacobian
- 8: Solve the equations of the Karush-Kuhn-Tucker (KKT) optimality conditions
- 9: **if** KKT condition not satisfied **then**
- 10:     GOTO 3
- 11: **else**
- 12:     STOP.
- 13: **end if**
- 14: **Output**: optimal state and control trajectories, optimal objective function value, number of iterations and CPU time.

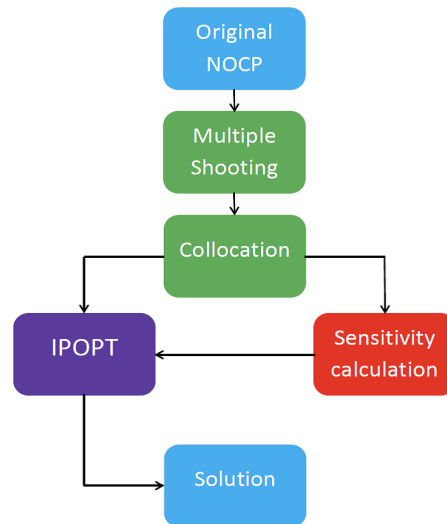


Figure 1: : Combined multiple shooting and collocation (CMSC) framework

Algorithm 1 is general for CMSC from the work of [17, 18]. This work implements a modified and improved version of Algorithm 1. Hence, Algorithm 2 provides a modified CMSC.

**Algorithm 2** : A modified CMSC

- 1: **Input:** Time horizon, number of subintervals, number of state and control variables, lower and upper bounds for the control and state variables, model equations, objective function, initial guesses, optimizer options, number of collocation points in subintervals, user-defined options, e.g., choice of a linear algebra algorithms, etc.
- 2: Calculate in symbolic form the Jacobian of the system and all directional derivatives.
- 3: Corresponding to the number of subintervals, states/controls, and collocation points, construct derivative matrices
- 4: Initialize and construct symbolic representations of the objective function and corresponding gradient.
- 5: Initialize and construct symbolic representation of the constraint functions and corresponding Jacobian.
- 6: Give an initial guess for the optimization variables
- 7: Set options for Ipopt to provide Hessian approximation, tolerance of the solution, and other user specified options.
- 8: Call the Ipopt solver
- 9: Call plotting routine and save the results.

The improvements provided in Algorithm 2 are:

- the symbolic pre-calculation to obtain representations for the values of the objective function, constraints, and corresponding sensitivities (Jacobians and derivatives),
- facilitate the use of several linear algebra algorithms to give the user a choice of options and improve the efficiency of computations,
- implementation on JModelica.org platform for a wider public accessibility.

Section 4 discusses the software implementation of Algorithm 2.

## 4 Framework and software components

In the framework shown in Fig. 2, the NOCP is modeled under JModelica.org using a Python script. Then the problem is discretized using our CMSC with the help of CasADi to obtain an NLP. Subsequently, CasADi is again invoked to generate symbolic expressions for the derivatives. Finally, JModelica.org

invokes Ipopt to solve the NLP by using the pre-calculations. All problem definitions and our custom implementations are done using the Python programming language.

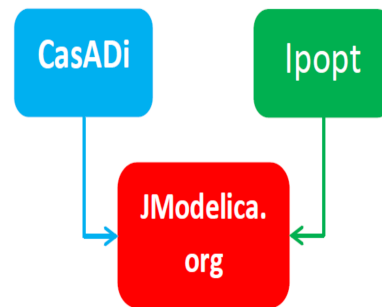


Figure 2: : Software framework

In the next subsections, we give a brief review on the software components.

### 4.1 JModelica.org

JModelica.org [2] is a Modelica-based open source software platform for modeling and solving optimal control problems and implementation of user-developed algorithms. JModelica.org was first developed at the Department of Automatic Control, Lund University. Currently, it enjoys active support from the industry (Modelon AB) as well as academic and research institutions. Since JModelica.org is extensible through user-designed algorithms, we have decided to implement our algorithm under this framework.

Among the salient features of JModelica.org are:

- support for mixed-language programming mode,
- easy and smoother integrability of custom numerical libraries,
- support for object-oriented modeling based on Modelica,
- wider public access owing to simpler user interfaces.

### 4.2 CasADi

CasADi is an open source software library for symbolic automatic differentiation of functions [5]. CasADi uses computer algebra techniques to implement the forward and adjoint automatic differentiation techniques and facilitate the pre-computation of gradients and Jacobian of objective functions and constraints, respectively.

One of the major features of CasADi is its high integrability to widely available open source numerical libraries and optimization solvers. CasADi is written using C++ programming language. However, it can be interfaced to numerical solvers based on various programming language implementations, e.g., C, C++, Python, FORTRAN, etc. The recent integration of CasADi to the JModelica.org platform [4] is one proof for its high integrability and applicability for the solution of complex optimal control problems. Therefore, in our implementation, we have exploited the potential of CasADi for symbolic pre-computation of derivatives in order to improve the efficiency of computations.

### 4.3 Ipopt

Ipopt is a software implementation of an interior point algorithm coupled with a filter line-search algorithm [19]. Ipopt is a state-of-the-art solver for large-scale NLP problems. Consequently, it facilitates the efficient solution of large-scale optimal control problems using the "first discretize, then optimize" approach.

In general, Ipopt can be used to solve NLP problems of the form

$$\min_{x \in \mathbb{R}^n} f(x) \quad (20)$$

subject to :

$$g_{min} \leq g(x) \leq g_{max}, \quad (21)$$

$$x_{min} \leq x \leq x_{max}, \quad (22)$$

where  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  is the objective function, and  $g(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is the constraint function. The vectors  $x_{min}$  and  $x_{max}$  are the bounds on the variables  $x$ , and the vectors  $g_{min}$  and  $g_{max}$  denote the lower and upper bounds on the constrained function  $g(x)$ . Furthermore, equality constraints can also be stated in the above formulation by setting the corresponding components of  $g_{min}$  and  $g_{max}$  to the same value. The functions  $f(x)$  and  $g(x)$  can be nonlinear and non-convex. Theoretically,  $f(x)$  and  $g(x)$  are required to be twice continuously differentiable. However, Ipopt is capable of working with first order information, so that Hessian matrices can be approximated numerically.

## 5 Case studies

To investigate the performance of the modified method, the following two case studies are considered.

### 5.1 A nonlinear batch-reactor

The system has two state variables  $x_1(t)$  and  $x_2(t)$  (corresponding to concentrations of two species) and one control variable  $u(t)$  (reaction temperature). The objective of this benchmark problem is to achieve a maximum product output of  $x_2(t_f)$ . The NOCP is formulated as follows:

$$\max_{u(t)} x_2(t_f) \quad (23)$$

subject to :

$$\dot{x}_1(t) = - \left( u(t) + \frac{u^2(t)}{2} \right) \cdot x_1(t), \quad (24)$$

$$\dot{x}_2(t) = u(t) \cdot x_1(t), \quad (25)$$

$$x_1(t_0) = 1, \quad (26)$$

$$x_2(t_0) = 0, \quad (27)$$

$$0 \leq \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} \leq 1, \quad (28)$$

$$0 \leq u(t) \leq 5, \quad (29)$$

$$0 \leq t \leq 1. \quad (30)$$

The objective function in equation (23) describes the amount of output of the second species at the final time. This example has been also studied in [6, 14, 17, 18].

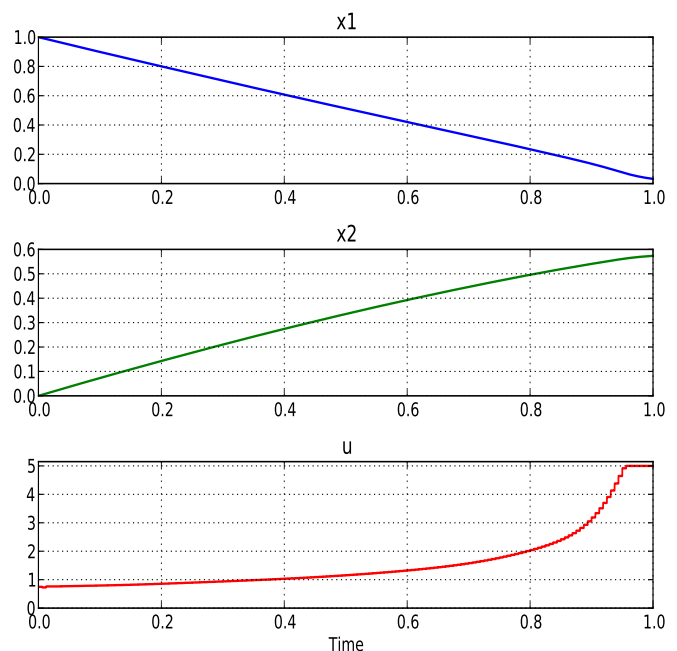


Figure 3: : Optimal solution of the NOCP (23) - (30)

The results shown in the Fig. 3 is obtained by using Algorithm 2 with 160 subintervals. In the next section we will present a comparative analysis of results

obtained from our Algorithm 2 and similar solution methods of other authors.

## 5.2 A satellite control problem

A nonlinear optimal control problem of a rigid satellite initially undergoing a tumbling motion is considered. The problem data are given as follows [17]:

- $I_1 = 10^6, I_2 = 833333, I_3 = 916677$  represent principal moments of inertia, respectively
- $T_{1s} = 550, T_{2s} = 50, T_{3s} = 550$  are time constants
- Initial states:  $x_1(0) = 0, x_2(0) = 0, x_3(0) = 0, x_4(0) = 1, x_5(0) = 0.01, x_6(0) = 0.005, x_7(0) = 0.001$
- Time horizon:  $[t_0, t_f] = [0, 100]$
- Fixed terminal state:  
 $x_{t_f}^\top = (0.70106, 0.0923, 0.56098, 0.43047, 0, 0, 0)$

The aim of the optimal control is to determine the torques that bring the satellite to rest in the specified time  $t_f = 100$ , while minimizing the performance index. The NOCP is defined as follows [17]:

$$\min_{u(t)} \left\{ \|x(t_f) - x_f\|^2 + \frac{1}{2} \int_{t_0}^{t_f} \|u\|^2 dt \right\} \quad (31)$$

$$\text{subject to: } \dot{x}_1 = \frac{1}{2} (x_5 x_4 - x_6 x_3 + x_7 x_2) \quad (32)$$

$$\dot{x}_2 = \frac{1}{2} (x_5 x_3 + x_6 x_4 - x_7 x_1) \quad (33)$$

$$\dot{x}_3 = \frac{1}{2} (-x_5 x_2 + x_6 x_1 - x_7 x_4) \quad (34)$$

$$\dot{x}_4 = -\frac{1}{2} (x_5 x_1 + x_6 x_2 + x_7 x_3) \quad (35)$$

$$\dot{x}_5 = \frac{(I_2 - I_3) x_6 x_7 + T_{1s} u_1}{I_1} \quad (36)$$

$$\dot{x}_6 = \frac{(I_3 - I_1) x_7 x_5 + T_{2s} u_2}{I_2} \quad (37)$$

$$\dot{x}_7 = \frac{(I_1 - I_2) x_5 x_6 + T_{3s} u_3}{I_3}. \quad (38)$$

Here  $x^\top(t) = (x_1(t), \dots, x_7(t))$  is the state vector and  $u^\top(t) = (u_1(t), u_2(t), u_3(t))$  is the control vector of the torques acting for the respective body-principle axes. The model equations (32) - (35) are the kinematic equations associated to the orientation and (36) - (38) are the dynamic equations associated to the motion of the satellite. The state variables  $x_1(t) - x_4(t)$  are the Euler parameters and  $x_5(t) - x_7(t)$  are the angular rates.

To solve this problem using Algorithm 2, we divide the time horizon into 50 subintervals and use a 3-point

collocation on each subinterval. This leads to an NLP with 507 variables and 357 constraints. The results of our implementation are depicted in Figs. 4 - 6.

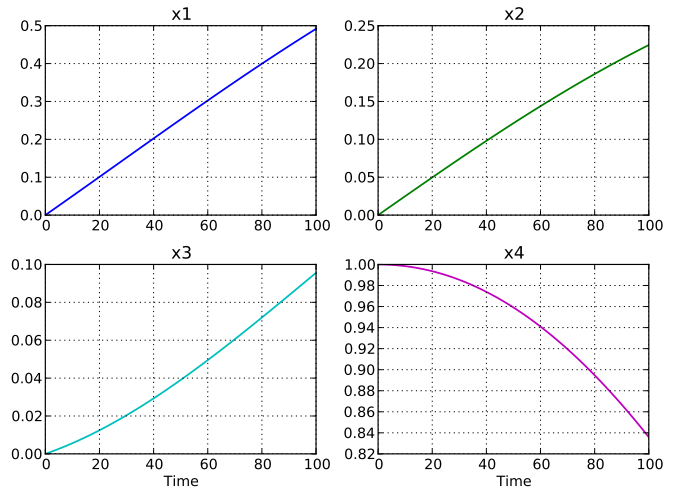


Figure 4: : Optimal states  $x_1 - x_4$

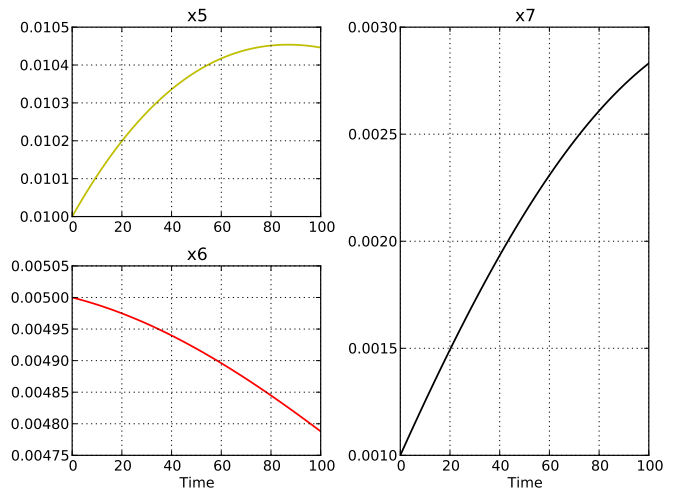
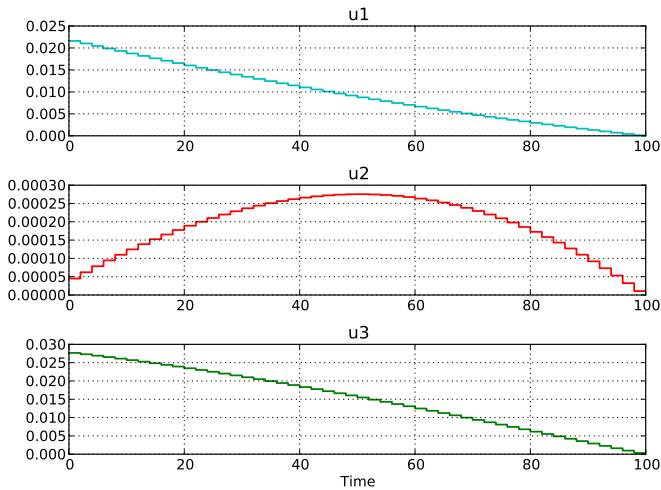


Figure 5: : Optimal states  $x_5 - x_7$

It takes 291 milliseconds of CPU time for the computation and the obtained optimal value of the objective function is 0.463968. In contrast, the optimal value of the objective function in [17] is 0.468287 obtained in 531 milliseconds of CPU time.

## 6 Comparative analysis

In this section we use the problem (23) - (30) for the purpose of comparison. Tables 1 - 4 present results obtained from similar, but different optimization methods. Note, that these results have been obtained using different computational platforms. Nevertheless, some ideas can be gained by observing the results obtained.


 Figure 6: : Optimal controls  $u_1(t), u_2(t), u_3(t)$ 

However, our modified CMSC algorithm is comparable to the collocation algorithm suggested in [15], since the implementation in [15] and of our modified CMSC are both done on the same type of computer with 3.2 GHz CPU frequency.

Table 1: CPU time (in milliseconds)

Number of subintervals	Modified CMSC	CMSC [17]	Collocation [15]	Multiple Shooting [14]	
				Unc., Sp.	Con., Co.
5	4	188	4	4	3
10	6	290	8	6	5
20	8	350	12	9	9
40	12	480	20	12	17
80	24	547	48	24	57
160	40	735	102	58	341
320	81	NaN	268	159	2717

In Tables 1 and 2

- Unc.: uncondensed SQP,
- Con.: condensed SQP,
- Sp.: Block structured QP solver using MA57,
- Co.: Matrix condensing and dense QP solver qpOASES [11].

Table 2: Number of iterations

Number of subintervals	Modified CMSC	CMSC [17]	Collocation [15]	Multiple Shooting [14]	
				Unc., Sp.	Con., Co.
5	14	16	17	7	7
10	16	21	24	9	9
20	19	21	21	9	9
40	22	25	20	10	10
80	27	23	23	10	11
160	18	23	23	12	12
320	14	27	27	12	14

As shown in Tables 1 and 2, the modified CMSC method performs more efficient, both in terms of CPU time and number of iterations, as compared to pure simultaneous (collocation) method. This facilitates the future use of Algorithm 2 for the model predictive control scheme on longer prediction horizons.

Considering Table 3, for the discretization using 5 subintervals, the collocation scheme of [15] provides lower function values as compared to Algorithm 2, but the CPU time of the modified CMSC method is better than the one reported in [17].

Table 3: Comparative results from discretization (5 subintervals)

	Objective function	Number of optimization variables	Number of constraints
Modified CMSC	0.56817	17	12
CMSC by [17]	0.56817	18	12
Collocation algorithm by [15]	0.57302	101	86
Multiple shooting by [14]	0.56838	18	10

Table 4: Comparative results from discretization (160 subintervals)

	Objective function	Number of optimization variables	Number of constraints
Modified CMSC	0.57354	482	322
CMSC by [17]	0.57354	483	322
Collocation algorithm by [15]	0.57354	3046	2566
Multiple shooting by [14]	0.57354	483	320

As shown in Table 4, our Algorithm 2 shows a higher performance in terms of CPU time as compared to all presented algorithms, still obtaining the the same objective function value as reported by other authors.

## 7 Conclusion and future work

This paper presents the first prototype of a modified combined multiple shooting and collocation method. The major difference from the original version of the CMSC approach in [17, 18] consists in using pre-calculated derivatives and their symbolic representation. That is, in every iteration of the optimization algorithm, sensitivities are automatically available without further calculations. The optimizer is provided with symbolic derivatives which are to be evaluated at the given iterate. This leads to accurate results with speedup of the overall computation time.

This preliminary investigation shows that the implemented algorithm has a competitive performance compared to other similar investigations. There is also a potential for parallel implementation of the proposed algorithm, whereby constraints to be considered on the coefficients of the collocation polynomials inside the shooting intervals. Furthermore, the modified CMSC method will be implemented to work directly with Modelica models along with a choice of local and global nonlinear equation solvers. Hence, the proposed framework will be refined to make it highly transparent, so that end-users can solve various types of applied optimal control problems. In addition, the algorithm will be extended to handle nonlinear model predictive control (NMPC) problems.

## 8 Acknowledgements

This work has been supported by Model Driven Physical Systems Operation project (MODRIO) by ITEA2, No. 11004, and by the German BMBF (BMBF Förderkennzeichen: 01IS12022H).

## References

- [1] Åkesson, J.: Optimica—An Extension of Modelica Supporting Dynamic Optimization. 6th International Modelica Conference, March 3 - 4, 2008, Bielefeld, Germany, pp. 57-66.
- [2] Åkesson, J., Årzén, K.-E., Gåfvert, M., Bergdahl, T., Tummescheit, H.: Modeling and optimization with Optimica and JModelica.org—Languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(2010)11, pp. 1737-1749.
- [3] Andersson, J., et. al.: Dynamic optimization with CasADi. In *Proceedings of the 51st IEEE Conference on Decision and Control*, Maui (USA), 2012.
- [4] J. Andersson, J., Casella, F., Diehl, M.: Integration of CasADi and JModelica.org. 8th International Modelica Conference, Dresden, 2011. DOI:10.3384/ecp1106321
- [5] Andersson, J., Åkesson, J., Diehl, M.: CasADi: A Symbolic Package for Automatic Differentiation and Optimal Control. *Lecture Notes in Computational Science and Engineering*, Vol. 87, Springer, 2012, pp. 297-307.
- [6] Bachmann, B., Ochel, L., Ruge, V., Gebremedhin, M., Fritzson, P., Nezhadali, V., Eriksson, L., Sivertsson, M.: Parallel Multiple-Shooting and Collocation Optimization with OpenModelica. *Proceedings of the 9th International Modelica Conference*, Munich, pp. 659-668, 2012.
- [7] Bartl, M., Li, P., Biegler, L. T.: Improvement of state profile accuracy in nonlinear dynamic optimization with the quasi-sequential approach. *AIChE Journal*, 57(2011)8, pp. 2185-2197.
- [8] Biegler, L. T. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. SIAM, 2010.
- [9] Bock, H. G., Plitt, K. J. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems, Prepr. 9th IFAC World Congress, Budapest, 1984, pp. 242-247.
- [10] Cuthrell, J. E., Biegler, L. T.: Simultaneous optimization and solution methods for batch reactor control profiles. *Comput. Chem. Eng.* 13(1989), pp. 49-62.
- [11] Ferreau, H. J.: Model predictive control algorithms for applications with millisecond timescales. PhD Thesis, KU Leuven, 2011.
- [12] Franke, R.: Formulation of dynamic optimization problems using Modelica and their efficient solution. 2nd International Modelica Conference, March 18 - 19, DLR, Oberpfaffenhofen, Germany, pp. 315 - 323.
- [13] Hong, W., Wang, S., Li, P., Wozny, G., Biegler, L. T.: A quasi-sequential approach to large-scale dynamic optimization problems. *AIChE Journal*, 52(2006)1, pp. 255-268.
- [14] Kirches, C., Wirsching, L., Bock, H. G., Schlöder, J. P.: Efficient Direct Multiple Shooting for Nonlinear Model Predictive Control on Long Horizons. *J. Process Control*, 22(2012), pp. 540-550.
- [15] Magnusson, F.: Collocation Methods in JModelica.org. Master Thesis, Lund University, February 2012.
- [16] Russel, R. D., Shampine, L. F.: A Collocation Method for Boundary Value Problems, *Numerical Mathematics*, 19(1971), pp. 1-28, Springer.
- [17] Tamimi, J.: Development of the Efficient Algorithms for Model Predictive Control of Fast Systems. PhD Thesis, Technische Universität Ilmenau, VDI Verlag, 2011.
- [18] Tamimi, J., Li, P.: A Combined approach to nonlinear model predictive control of fast systems. *J. Process Control*, 20(2010)9, pp. 1092-1102.
- [19] Wächter, A., Biegler, L.T.: On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, *Mathematical Programming, Ser. A*, 106(2006)1, pp. 25-57.

# DOML - a Compiler Environment for Dynamic Optimization Supporting Multiple Solvers\*

Tomasz Tarnawski   Radosław Pytlak

Warsaw Technical University, Institute of Automatic Control and Robotics  
ul. św Andrzeja Boboli 8, 02-525 Warsaw

## Abstract

The Modelica language may serve well as a base for defining optimal control problems, given a few relatively minor syntax extensions. One example proving that point is Optimica and another one is DOML (Dynamic Optimization Modeling Language) – installed on IDOS (Interactive Dynamic Optimization Language) and described in this paper. The DOML implementation is, actually, heavily based on the (open source) compiler of Optimica but it provides a number of important features absent in its precursor. One main extension of the compiler lies in its built-in mechanism supporting the use of many different optimization solvers (selected on the fly, depending on the content of the problem definition) and to seamlessly add new, external, solvers. In result, the range of problems that may be specified with DOML and solved in the IDOS environment is quite wide and keeps growing. The scope of problems ranges from some static optimization problems through regular ODE, parametric optimization, minimum time problems, up to DAE with higher index. DOML language extensions also provide preliminary support for multi-objective optimization and PDE problems.

*Keywords:* dynamic optimization; optimal control; Optimica

## 1 Introduction

Dynamic Optimization Modeling Language (DOML) was initially proposed ([23]) as a programming-language-independent communication format for newly developed Interactive Dynamic Optimization Server (IDOS, described e.g. in [22]). The IDOS server, envisioned by its

proponents as "the NEOS for optimal control", provides an on-line environment for dynamic optimization. It receives inputs — definitions of dynamic optimization problems specified in DOML — from its users and responds with solutions (if) found. Although still under development, IDOS is already operational, on-line at [32]. The main mode of working with the server is through a web browser, where the (logged-in) user may type problem definitions in DOML, submit tasks, view results etc. The elements and functions of the IDOS server, including exemplary computational results, have already been described in a few publications (e.g. in [22], [23] and [24]) and, in particular, in a parallel paper to be presented on the very same 10<sup>th</sup> Modelica Conference ([25]). Hence, here the functioning of IDOS itself will not be discussed any further, the core focus goes to the DOML format and its implemented compiler (as installed on IDOS).

The DOML format provides a mean for defining optimal control problems in a slightly extended Modelica syntax. In that respect it is very similar to Optimica (initially proposed in [2]) and in fact implementation of its compiler<sup>1</sup> is based on the (open source) Optimica compiler, available at [31]. Since Optimica itself has been presented in numerous papers, in particular on the Modelica Conference (e.g. [3]), it is not discussed here in much detail. Instead, the focus lies on the main features of DOML that set it apart from Optimica.

As stated above, DOML compiler is installed on a on-line-accessible server ([32]) and is used to interpret the input form (remote) users entering problems they want solved. In contrast, Optimica compiler comes as a part of a downloadable and locally installed JModelica.org environment, where the user is mainly exposed to Python

\*The work presented in the paper has been partially supported by NCBiR grants: R02-0009-06, PBS1/A7/6/2012

<sup>1</sup>DOML compiler is developed as open source. Contact the authors for setting up access to code repository

programming console as a mode of working with compiled problems, numerical packages and optimization solvers (see e.g. [4]). That architectural contrast dictates the main difference between the two compilers: DOML compiler provides inherent, built-in support for generating code for a multitude of different solvers and numerical packages (and new solvers may still be added to the environment) while in JModelica.org environment implementing any extensions or different optimization algorithms is (only) possible through Python scripts (see [26] for an example) – executed *on top* of the Optimica compiler.

One design principle for the IDOS server was its ease of use, particularly for users with limited programming skills. Hence, it seemed highly desirable to avoid any meanders on the road from a problem to its solution, such as the need of using other (programming) languages, learning environment’s internal API or studying interfaces to numerical libraries. In effect, all that an IDOS user must provide is the problem definition in DOML and DOML is then the only language that one really needs to comprehend in order to use the server. (However, as discussed later, an advanced user with good acquaintance with the compiler’s and server’s internals has also the ability to extend the server’s functionality and to implement own solving algorithms).

We find the need for implementing support for multiple solvers to be justified and intuitive. For once, dynamic optimization problems come in many different kinds (e.g. parametric, minimum time, DAE with higher index, etc) and so one solver would not be general enough to solve all of them efficiently and accurately; solvers dedicated to particular problem kinds simply do the job better. The second reason is to open the possibility for applying "solver chaining" i.e. a strategy of using two (or more) solvers consecutively on one problem. The first one yields a crude approximation while the next guarantees a more accurate solution but requires a reasonably good initial guess (and possibly other warm-starting information) – that may be taken from the first solver. Automation of such procedure seems very tempting, yet it poses a number of challenges and in some cases may plainly be impossible (e.g. due to the nature of information needed by some second-line solvers, such as [17]). Currently, a user must recode the problem manually through editing in the results

obtained from the first solver and resubmit the file. For more details on chaining solvers in IDOS see e.g. [24].

The paper is organized as follows. The following section 2 provides a general introduction to the DOML compiler. Section 3 proceeds towards the technical aspects dealing with the paper’s key issue — implementation of multi-solver support. Additional attention is devoted to the more advanced topic of supplementing the compiler with external code generators (section 3.1). Next, in section 4, the set of currently deployed solvers is briefly presented, together with (selected) code generators within the compiler implemented to couple with these solvers. Finally, section 5 provides supplementary information on other important features proposed in DOML (and setting it apart from Optimica) yet less tightly related to multi-solver support.

## 2 Compiler Overview

On the most general level, the working of DOML compiler is identical to that of Optimica. It reads in an input `.mo` file into memory, where it creates own representation of all elements of the problem’s definition. Based on that, it generates output file(s) written in a regular programming language (mostly C/C++) — therefore the compiler, formally, is actually more of a translator (from Modelica to C/C++). The produced files may then be placed as input to the standard (`gcc`) compiler and linker (bringing in required external libraries) to produce an executable file that will carry out the actual solving algorithm.

The generation of C (C++) code is template-driven: in order to produce any output, the compiler reads in an external *template* file – resembling closely a regular C++ file where only certain spots are specially marked (with so called *tags*). These tags represent places in the code where external snippets of code are to be pasted, e.g. the `$n_real_x$` tag will be replaced with the number of state variables defined in the model. The code generating method reads in the template file(s) and reproduces it to the output. Every time it comes across a tag, however, it replaces it in the output with an appropriately elaborated fragment of code — depending on the tag it may be a single number literal or several lines of C/C++ code. A special mechanism dispatches the calls to the right



code (or: snippet) generating methods based on the textual content of the tag.

The compiler is fully implemented in Java. Its executable, *together with all required template files*, is packed into a single Java jar file: `DOMLCompiler.jar` — which is where the differences against Optimica start. Typical execution of the compiler, on an exemplary input file `theInput.mo`, would take the form of:

```
> java -jar DOMLCompiler.jar theInput.mo
-o outDir
```

and all produced output would end up in `outDir` directory. It proved very convenient to supplement the set of templates with a `Makefile` so that in the output folder, next to C/C++ source code (and possibly other auxiliary) files, a `Makefile` would also be present. Then, simply running

```
> make
```

in that directory produces the executable to run and solve the problem.

The set of generated files constitutes, essentially, a problem-specific implementation of a dynamic optimization algorithm and most usually it makes an extensive use of (external) numerical libraries — through their APIs. Naturally, the generated files (and, in particular, the build commands in `Makefile`) must closely correspond with the libraries installed on the computer. As the compiler provides the problem-specific information, its back-end — the code generating *engine* — becomes responsible for "mating" the compiler with the solver(s). The two *pistons* of that engine are: the collection of template files (mentioned above) and the Java class implementing the logic for producing code snippets for each tag. Then their common *crankshaft*, to keep with the comparison, would be the set of tags present in the templates and rendered by the Java class (technically, it is in fact a fair number of small internal classes defined within the main code generator class; each of them is responsible for interpreting its separate tag).

The design of the DOML compiler provides for an easy way for adding and/or replacing back-ends. Such back-end block, composed of a code generating class together with its related templates, can therefore be viewed as a plug-in to the compiler (especially, since it can be packaged into a completely separate jar file, as described in detail in section 3.1). The concrete code generator is selected and loaded on the fly, based on the con-

tent of the input `.mo` file. Irrespective of the code generator eventually employed, the mode of executing problems under DOML is always the same: first run `jar`, then `make`.

In that respect, DOML is very different from Optimica which contains only built-in (and "hard-coded") code generators — to C and to XML. The C generator targets a single, specific optimal control solver environment (contained within JModelica.org environment). Admittedly, the XML code generator grants the user more flexibility, as the produced files may be imported by other tools (as discussed in [1]). The flipside of that approach is that it burdens the user with handling yet another file format (XML, this time) that he or she has to transport from one environment to another — either manually or through Python scripting (not mentioning, that such additional tools would then also need to be installed). This may be perfectly fine for users proficient with programming, scripting and managing multiple tools installed on their computers, but not all of them are. As already indicated, IDOS targets different users — ones, that would prefer not to (learn and) do all that, and instead to have one simple tool (a web browser) and one language (DOML) for solving all their (optimal control) problems.

### 3 Implementation of Multiple-solver support

In the context of DOML compiler, the term *solver* refers to a set of templates together with numerical libraries used by them and needed to build the executable. In other words, each solver is an implementation of a particular dynamic optimization algorithm; as soon as it is filled with problem specific data it may be compiled and executed. The actual collection of solver packages deployed on IDOS server (and required libraries installed alongside) is discussed in [25] and here touched upon in section 4. For the purpose of the current discussion, it is sufficient to recognize that there may be present quite a few different packages implementing algorithms for solving distinct kinds of optimization problems and with time still new packages may be implemented and plugged-in to the environment.

In DOML compiler environment each solver's code generating class — i.e. the class containing the logic to fill up all templates for particular

```

package pl.pw.DOML.codegen;

public abstract class BaseDOMLGen
    extends OptimicaCGenerator {

    public static BaseDOMLGen
        loadGenerator(String solverName,
                      FDOMLClass fc,
                      Properties props) {
        ...
    }

    public abstract ArrayList<String>
        getTemplateFNames();

    public void
        generate(String templateF,
                 String outF) { ... }

    ...
}

```

Listing 1: BaseDOMLGen class excerpt

solver — must be a subclass of BaseDOMLGen. Excerpts of this class, shown on listing 1, present its most important features. It:

- implements static method `loadGenerator()` for searching, loading and instantiating the appropriate, concrete code generating class. It utilizes Java reflection mechanism to look for proper compiled Java classes.
- declares `getTemplateFNames()` abstract method that every generator must override to communicate (through call-back) its set of template file names to the compiler. This way, each code generator may be tied to its own, completely separate set of templates.
- implements `generate()` method that carries out the process for each template file (whose name is provided as the first parameter) by reading it and dispatching calls to individual tag-generating methods.

Any code generator class derived from BaseDOMLGen needs to implement only the `getTemplateFNames()` method and the logic of generating code snippets for its specific tags (i.e.: tags present in templates associated specifically with that generator).

The class searching and loading mechanism relies on the use of Java annotation `@Solver` defined

specifically for that purpose. Each code generator implemented according to the description above must be appropriately annotated, where at least the solver's unique name must be given — as on the example below with "olado" :

```

@Solver("olado")
public class DOMLGen4Olado
    extends BaseDOMLGen { ...

```

All template files used by that solver must then reside in the jar file, inside its `/templates/olado` subdirectory.

The annotation `@Solver` also provides means for specifying the kinds of problems that a given solver package can handle, through a set of attributes defining whether the package requires (TRUE), accepts (DONT\_CARE) or disallows (FALSE) certain qualities and elements of a problem. The default value for all `@Solver`'s attributes is FALSE, hence the ones not applicable may be omitted. Example below illustrates the point:

```

@Solver(value = "rkcon",
        freeFinalTime
            = ApplicableTo.TRUE,
        minFinalTime
            = ApplicableTo.DONT_CARE,
        residualEquations
            = ApplicableTo.DONT_CARE,
        finalStateConstraints
            = ApplicableTo.DONT_CARE)
public class DOMLGen4RKcon
    extends BaseDOMLGen {

```

As of now, this annotation's attributes describe, whether the given solver is applicable to:

- with respect to the problem's structure — static optimization, ODE or DAE systems, parametric optimization (in particular: optimization of the system's initial state), PDE problems;
- with respect to the elements present in the definition — problems with: free final time (in particular, where final time is the objective), equations specified in residual form, integer-valued decision variables, multiple objectives;
- with respect to the kinds of constraints defined — box constraints, constraints on the final state, general functional constraints or non-stationary constraints.

This setup should not yet be considered final but it is well past the proof-of-concept stage. Having the above list as a rough guidance, the goal of development efforts of the IDOS server is to furnish a host of solvers covering typical cases of dynamic optimization problems. The current state of efforts, as far as code generators implemented and the set of corresponding libraries installed, is briefly discussed in section 4.

When executing a problem specified in DOML, one may force the use of a particular package through (Modelica-like) annotation `solver` handled by the DOML compiler, as so:

```
annotation(solver="olado");
```

If such annotation is omitted in the DOML input (or `solver` is specified as "auto"), the compiler makes its best effort to load the generator class that fits the current problem. Upon entering the code generation stage (i.e. after parsing input file, doing semantic analysis and problem transformations e.g flattening or alias elimination) the compiler queries the object containing the AST representation of the input — to classify the problem along the lines sketched above. Then, it browses available code generator classes (subclasses of `BaseDOMLGen`) and tries to match their `@Solver` description to that classification. If an exact counterpart cannot be found, a simple (preliminary) heuristic is used to choose the closest match. An object of the selected class is instantiated and used thereupon.

### 3.1 External code generators

In the mechanism described above, the logic responsible for browsing for available code generator classes implements one additional, powerful feature. The searching is not limited to the content of the `DOMLCompiler.jar` file, but in fact incorporates all jar files found in the current folder. This way any external solver, packaged (together with its templates) into a jar file, may equally well be used by the compiler task. An advanced user may therefore implement own solver jar package (which is the harder part) and add it seamlessly and non-intrusively to the compiler (which then is as easy as uploading the jar file on the server).

When implementing own solver package, say under the name `mySolverPack`, one must produce a Java jar file that fulfills the following minimum requirements:

- it must contain the code generator class — subclass of `BaseDOMLGen` — annotated as `@Solver("mySolverPack")` and implementing `getTemplateNameNames()` that returns a collection of names of solver's template files (including a makefile);
- it must contain all template files placed in `/templates/mySolverPack` subfolder;
- templates may use only numeric libraries installed on the server (installing extra libraries on user accounts is not yet supported), the makefile should be written accordingly;
- the code generating class must handle appropriately solver-specific tags appearing in the templates.

Preferably, the annotation `@Solver` should also specify applicability of that package (as discussed earlier); otherwise, one will be able to use the solver only by specifically requesting its name through annotation in the DOML input file.

The above requirements clearly imply, that this functionality is meant for advanced users—developers of optimization algorithms. Preparing the set of templates calls for a good knowledge (and at least some practice of use) of libraries installed on the server while implementing own code generator class is possible only with good familiarity of the compiler's internal API (in particular: the data structure representing the compiled input problem).

## 4 Implemented solvers and code generators

At the moment the IDOS server can handle control problems described with ordinary equations and differential–algebraic equations but the incorporation into the IDOS solvers for problems with partial differential equations is also under way.

The IDOS server enables solving optimal control problems by using essentially different methods ranging from ones based on *a priori* discretization through utilizing variable stepsize integration and adjoint equations to shooting methods applied to differential equations derived from necessary optimality conditions. These methods and solvers are discussed in some detail in [25].

Obviously, not all solvers can be used on every problem. The compiler is fit to cooperate with

the installed solvers, through the following code generator classes implemented in DOML compiler:

- `DOMLGen40lado` — ODEs are discretized *a priori* and then the large-scale NLP problem solved by a static optimization solver (Ipopt [29], KNITRO, etc)—the library OLADO is described in [5] and uses several interior point methods ([16],[11]) for QP problems to which solvers refer;
- `DOMLGen4cvodes` — ODEs treated as continuous time (adaptive stepsize integration using SUNDIALS, [15]), the optimization problem solved with the help of adjoint equations (in the reduced space) and the Ipopt package—the solver is based on cvodes procedure ([28]) from SUNDIALS;
- `DOMLGen4bndsc0` — indirect shooting method using the BNDSO package developed by Oberle ([17]) ;
- `DOMLGen4Pantelides` — higher index DAEs, implementing the Pantelides' graph based initialization algorithm ([19]), also Maxima package for symbolic differentiation is used ([30]);
- `DOMLGen40ladoBonmin` — ODEs with integer valued controls, solved through *a priori* discretization of ODEs as applied in the OLADO package and with the help of MIP solvers from BONMIN package ([6],[7]);
- `DOMLGen4cvodesBonmin` — ODEs with integer valued controls, the optimization problem solved with the help of adjoint equations (evaluated by SUNDIALS programs) and BONMIN package;
- `DOMLGen40ladoHqp0muses` — various dynamic optimization solvers based on HQP and OMUSES packages ([9],[10]);
- `DOMLGen4RKon` — index one DAEs treated by implicit Runge–Kutta scheme, optimization problem solved in reduced space by RKCON solver (special treatment of state constraints)—the justification for the solver RKCON is presented in [20];
- `DOMLGen4Radau` — up to index three DAEs treated by RADAU5 (implicit R-K) solver ([14],[12],[13]), optimization problem solved

in reduced space by RKCON solver, the package is based on adjoint equations for higher index DAEs derived in [21].

The above list is not exhaustive as, for instance, a few of other back-ends undergo testing and/or are under construction — e.g. `DOMLGen4PDE` or `DOMLGen4staticIpopt` (where names imply their purpose).

## 5 Other enhancements in DOML

The main direction of extending DOML syntax revolved around the idea of "chaining" of solvers, i.e. using a number of solvers in a sequence so that a succeeding solver would (use as an initial guess and) improve on the solution arrived at by its predecessor. Usually an approximate solver would be used first, followed by one that is more accurate yet also more sensitive (susceptible) to the choice of starting point.

One of DOML language proposed features is an external package added into the Modelica Standard Library — `Modelica.DOML` — with its predefined (/built-in) data types and functions. Currently `Modelica.DOML` contains two subpackages: `internal` and `Inputs`. Importantly, the definitions in `Modelica.DOML.internal` are automatically visible in any DOML file i.e. it is always implicitly imported by the compiler (therefore, specifying explicitly `import Modelica.DOML.internal.*`; would be redundant and will have no additional effect). The package `internal` contains, among others, the definition of `Formula` class needed for certain aspects of solver chaining (but also e.g. `Domain` class used in defining PDE problems). In turn, the purpose of `DOML.Inputs` was to allow for easy definition of known (predefined) signals, most notably with `Spline` – using tabular values to produce piece-wise constant (degree = 0) or continuous (degree  $\geq 1$ ) signal, based on polynomial interpolation.

A simple and intuitive language extension (or perhaps just a bug fix to `Optimica`) was to allow, in certain cases, *continuous* variability in the right-hand-side expression defining the value of the `initialGuess` attribute. If a variable's value can change over time, then it seems reasonable to allow the user to provide an initial guess for it that also is not constant over time. After that enhance-

```

Modelica.DOML.Inputs.Spline
  u_init(startTime=0, finalTime=1,
          table=[0,0; 0.2,.096;
                 0.4,.1; 0.6,.1;
                 0.8,.096; 1,0],
          degree = 1);
Real u(initialGuess = u_init.y);

```

Listing 2: Specifying on initial guess signal for a continuous variable

```

class Formula
  parameter Boolean linear;
  Real lagrange(dual = true);
  parameter Integer order;
end Formula;

```

Listing 3: Definition of the predefined class **Formula**.

ment, the above-mentioned `Spline` may be used to feed an arbitrary signal as an initial guess of a variable, as illustrated on listing 2. This way, a user can run optimization with a fairly good starting point, for instance obtained by pre-solving the problem with a different (simpler) solver or with coarser discretization — a typical implementation of the strategy of "chaining of solvers".

Next to the (near optimal) trajectories found for variables and controls, other (by)products of (pre)solving an optimal control problem may be available and beneficial when warm-starting a high-precision solver. Among such, are the values of adjoint variables and Lagrange multipliers arrived at near the solution. (Adjoint variables and Lagrange multipliers are, arguably, two names representing the same fundamental concept hence, for the sake of simplicity, for both cases the syntax provides the same name: `lagrange`). Assigning the `lagrange` values to equations (/constraints) needs a mechanism for identifying particular formulas. In DOML it is devised by means of labeling them, as so

```
eq_x1: der(x1) = p1*x1 - p2*x2*x3;
```

with simultaneously declaring the "label variable" as **Formula** `eq_x1`; . The compiler checks if all variables of type **Formula** are referenced as labels and, conversely, that all labels used next to equations are appropriately declared variables. The **Formula** class is declared in `DOML.internal`, as shown on listing 3.

The objects of the new predefined type **Formula**

are meant to be used to specify attributes describing the particular equation or constraint – most notably the above mentioned adjoint variable/Lagrange multiplier, but for prospective use other attributes were defined as well. Similarly to the earlier example, one can now specify an `initialGuess` for the lagrange attribute associated with a particular equation or constraint. Such additional information may be beneficial, or plainly required, in certain solvers (e.g. BNDSCO).

The last to discuss modification of DOML vs. Optimica lies in a slightly different way of defining the objective within an **optimization** class. DOML breaks up with Optimica's objective attribute and introduces two keywords, **minimize** and **maximize**, in its stead. Each has an almost identical meaning to Modelica's **parameter** keyword, but in addition signals that the given parameter holds the objective value.

The modification was motivated by its following advantages:

- it clearly indicates the direction of optimization; in some domains the default of minimization may not be as natural.
- objective variables can now have meaningful names;
- it becomes possible (and strikingly simple) to define multi-objectives problems in a natural way e.g. with:

```

maximize Real profit =... ;
minimize Real deliveryTime =... ;

```

(implementing a multi-objective solver is, naturally, quite another issue)
- it makes it possible to define problems with integer valued (or even Boolean valued, for that matter) objective functions (again, we are not touching on the implementation aspect here).

A number of other important language extensions is related to using DOML to specify and (prospectively) solve PDE problems. This aspect also deserves attention but is clearly beyond the scope (and page allowance) of this paper. This aspect of implementation is described in [23] and was partially inspired by [27].

## 6 Summary and future work

IDOS (Interactive Dynamic Optimization Server) provides an environment for defining and solving optimal control problems in DOML (Dynamic Optimization Modeling Language) — a modeling language derived from Modelica (/Optimica). The main focus of the paper was put on these features of the server's environment — in particular, of the DOML compiler — that provide support for applying different solvers to different problems. Additional information about the current range of implemented solvers was also provided.

A number of features proposed in DOML set it apart from Optimica. The authors believe, that these modifications deserve to be called "improvements" and/or "extensions" as they seem to enhance both expresiveness and applicability of the language. Their presentation and discussion is scattered throughout the paper and therefore, for clarity and readability, here we (re)state them all together in a concise manner. The new features of DOML are:

- implementation of multiple solvers with a mechanism for adding (plugging-in) new code generators (vs. hard-coded, closed set of code generators in Optimica);
- `annotation(solver)` allowing the user to choose a particular solver package to be used;
- support for continuous variability of the `initialGuess` attribute (vs. parameter variability in Optimica) adds flexibility in defining initial (warm-starting) point of computations, e.g. with using provided `Spline` signal generator;
- mechanism for labeling equations and constraints (missing in Modelica/Optimica);
- mechanism for referencing (and warm-starting) adjoint variables / Lagrange multipliers of (labelled) formulas;
- keywords `maximize` and `minimize` defining the objective function with indicating the direction of optimization (vs. `objective` attribute in Optimica)  
note: makes it possible to define multiple-objective problems, now the objective does not have to be real-valued.

Apart from the above list, the lion's share of the DOML compiler functionality is inherited directly from the (JModelica.org) Optimica compiler whose authors, naturally, deserve due recognition and honor.

Currently, significant parts of the server's functionality are still under development and testing. There is still much room for improvement, for instance as far as robustness (e.g. in cases of some less usual formulations that are acceptable from the language point of view but not recognized by code generating rules) or graceful error handling are concerned. A further reaching to-do list also contains: polishing the mechanism for problem classification and choosing solvers/generators (more advanced rules for matching most appropriate solver to a given problem) and better support for chaining of solvers (possibly, through producing solver's output in the form of "enriched" input, with solution pasted in as, e.g. the value of control's `initialGuess` attribute; such output could then be directly used to warm-start another solver).

One central goal in the design of the environment was to provide mechanisms for easily extending the compiler's code generating functionality in order to support a growing number of solver packages. As discussed in some detail above, the goal has already been reached successfully. In result, the compiler offers a remarkable feature: extensions of its code generating functionality is possible virtually on the fly — by simply adding an external jar file and without any interference with the compiler's executable. With this mechanism, remote users of IDOS may be enabled to implement own solver packages and deploy them on the server. If the proposed solution catches on and stands the test of time, IDOS could become an important, extensible, multi-user environment for solving wider and wider range of optimal control problems — "the NEOS of optimal control".

## References

- [1] Andersson J, Åkesson J, Casella F, Diehl M. Integration of CasADi and JModelica.org. In: Proceedings of the 8th Modelica Conference 2011, Dresden, Germany, Modelica Association, 20-22 March 2011.
- [2] Åkesson J. Tools and Languages for Optimization of Large-Scale Systems. Lund, Swe-

- den: *PhD thesis*, Department of Automatic Control, Lund University, 2007.
- [3] Åkesson J. Optimica—an extension of Modelica supporting dynamic optimization. In: Proceedings of the 6th Modelica Conference 2008, Bielefeld, Germany, Modelica Association, 3-4 March 2008.
- [4] Åkesson J, Gäfvert M, Tummescheit H. JModelica—an Open Source Platform for Optimization of Modelica Models. In: Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling, Vienna, Austria, TU Wien, February 2009.
- [5] Błaszczyk J, Karbowski A, K. Malinowski K. Object Object library of algorithms for dynamic optimization problems: benchmarking SQP and nonlinear interior point methods. In: Journal of Applied Mathematics and Computer Science, Vol. 17, 515–537, 2007.
- [6] Bonami P, Wachter A, Biegler L.T, Conn A.R, Cornujols G, Grossmann I.E, Laird C.D, Lee J, Lodi A, Margot F, Sawaya N. An algorithmic framework for convex mixed integer nonlinear programs. In: Discrete Optimization, Vol. 5, 186–204, 2008.
- [7] Bonami P, Lee J BONMIN Users' Manual, 2009.
- [8] Bonami P, Lodi A, Cornujols G, Margot F. A feasibility pump for mixed integer nonlinear programs. In: Mathematical Programming, Vol. 119, 331–352, 2009.
- [9] Franke R. Anwendung von Interior-Point-Methoden zur Lösung zeitdiskreter Optimalsteuerungsprobleme. *Master's thesis*, Technische Universität Ilmenau, Fakultät für Informatik und Automatisierung, Institut für Automatisierungs- und Systemtechnik Fachgebiet Dynamik und Simulation ökologischer Systeme, Ilmenau, Germany, October 1994 (in German).
- [10] Franke R. OMUSES — a Tool for the Optimization of MULTistage Systems and HQP — a Solver for Sparse Nonlinear Optimization. Dept. of Automation and Systems Engineering, Technical University of Ilmenau, Germany, September 1998.
- [11] Gondzio J. Multiple centrality corrections in a primal-dual method for linear programming. *Technical Report 1994.20*, Department of Management Studies, University of Geneva, Geneva, Switzerland, November 1994.
- [12] Hairer E, Lubich Ch, Roche M. The numerical solution of differential–algebraic equations by Runge–Kutta methods. In: Lecture Notes in Mathematics 1409, Springer–Verlag, Berlin, Heidelberg, 1989.
- [13] Hairer E, Norsett S.P, Wanner G. Solving Ordinary Differential Equations I., Springer–Verlag, Berlin Heidelberg New York, 2008.
- [14] Hairer E, Wanner G. Solving Ordinary Differential Equations II, Springer–Verlag, Berlin Heidelberg New York, 1996.
- [15] Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., Woodward, C.S. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, preprint, UCRL-JRNL-200037, 2004.
- [16] Mehrotra S. On the implementation of a primal–dual interior point method. In: SIAM Journal on Optimization, Vol. 2, 575–601, 1992.
- [17] Oberle H.J, Grimm W. BNDSCO – A Program for the Numerical Solution of Optimal Control Problems, Report No. 515 der DFVLR (German Test and Research Institute for Aviation and Space Flight), 1989.
- [18] Oberle H.J. Numerical Solution of Minimax Optimal Control Problems by Multiple Shooting Technique. In: J. Optimization Theory and Applications, Vol. 50, 331–364, 1986.
- [19] Pantelides C. Consistent initialization of differential–algebraic system. In: SIAM J. Scientific and Statistical Computation Vol. 9, 213–231, 1988.
- [20] Pytlak R. Numerical procedures for optimal control problems with state constraints. Lecture Notes in Mathematics 1707, Springer–Verlag, Heidelberg, 1999.

- [21] Pytlak R. Numerical procedure for optimal control of higher index DAEs. In: *J. Discrete and Continuous Dynamical Systems*, Vol. 29, 1–24, 2011.
- [22] Pytlak R, Tarnawski T, Fajdek B, Stachura M. Interactive dynamic optimization server—connecting one modeling language with many solvers. In: *Optimization Methods and Software*, 2013. <http://dx.doi.org/10.1080/10556788.2013.799159>.
- [23] Pytlak R (ed.). Interactive computer environment for solving optimal control problems—IDOS. *final Research Report, under the grant R02-0009-06*, Warsaw, Poland, 2012.
- [24] Pytlak R, Błaszczuk J, Krawczyk K, Tarnawski T. Solvers chaining in the IDOS server for dynamic optimization. In: *Proceedings of 52nd Conference on Decision and Control*, Florence, Italy, 10-13 December 2013.
- [25] Pytlak R, Tarnawski T. IDOS – (also) a Web Based Tool for calibrating Modelica model. Accepted for: 10th Modelica Conference 2014, Lund, Sweden, Modelica Association, 10-12 March 2014.
- [26] Rantil J, Åkesson J, Führer C, Gäfvert M. Multiple-Shooting Optimization using the JModelica.org Platform. In: *Proceedings of the 7th Modelica Conference 2009*, Como, Italy, Modelica Association, 20-22 September 2009.
- [27] Saldamli L. PDEModelica – A High-Level Language for Modeling with Partial Differential Equations, *PhD. thesis*, Department of Computer and Information Science, Linköping University, 2006.
- [28] Serban R, Hindmarsh A.C. CVODES, the sensitivity-enabled ODE solver in SUNDIALS. In: *Proceedings of the 5th International Conference on Multibody Systems, Nonlinear Dynamics and Control*, ASME, Long Beach, CA, 2005.
- [29] Wachter A, Biegler L.T. On the implementation of an interior point Line search filter algorithm for large scale nonlinear programming. In: *Math. Programming*, Vol. 106, 25–57, 2006.
- [30] Maxima Manual Ver 5.30.0, <http://maxima.sourceforge.net/docs/manual/en/maxima.pdf>, November 2013.
- [31] Open Source Project: JModelica.org. <http://www.jmodelica.org/>. November 2013.
- [32] Interactive Dynamic Optimization Server. <http://idos.mchtr.pw.edu.pl/>. November 2013.



# Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C

Vitalij Ruge   Willi Braun   Bernhard Bachmann  
{vitalij.ruge, willi.braun, bernhard.bachmann}@fh-bielefeld.de  
Bielefeld University of Applied Sciences, Department of Mathematics and Engineering,  
Bielefeld, Germany

Andrea Walther   Kshitij Kulshreshtha  
andrea.walther@uni-paderborn.de   kshitij@math.upb.de  
Universität Paderborn, Institut für Mathematik,  
Paderborn, Germany

## Abstract

Efficient calculation of the solutions of nonlinear optimal control problems (NOCPs) is becoming more and more important for today's control engineers. The systems to be controlled are typically described using differential-algebraic equations (DAEs), which can be conveniently formulated in Modelica. In addition, the corresponding optimization problem can be expressed using Optimica.

Solution algorithms based on collocation methods are highly suitable for discretizing the underlying dynamic model formulation. Thereafter, the corresponding discretized optimization problem can be solved, e.g. by the interior-point optimizer Ipopt. The performance of the optimizer heavily depends on the availability of derivative information for the underlying optimization problem. Typically, the gradient of the objective function, the Jacobian of the DAEs as well as the Hessian matrix of the corresponding Lagrangian formulation need to be determined. If only some or none of these derivatives are provided, usually numerical approximations are used by the optimizer internally.

OpenModelica supports the Optimica language and is capable of automatically generating the discretized optimization problem using collocation methods as well as the whole symbolic machinery available. In addition, all necessary derivative information is determined using the automatic differentiation capabilities of ADOL-C, which has now been integrated into the OpenModelica environment.

*Keywords:* Modelica; optimization; automatic differentiation; collocation; OpenModelica; ADOL-C

## 1 Introduction

The aim of this paper is to describe an efficient new solution process implemented in OpenModelica [11] for nonlinear optimal control problems. This effort continues the development of the collocation approach already discussed in [3], which has been successfully tested using the algorithmic differentiation tool CasADi [18]. Several enhancements, e.g. special treatment of the first collocation interval, integration of the automatic differentiation tool ADOL-C, as well as efficient and stable calculation of all derivative information, have been realized in OpenModelica and are demonstrated within this paper.

Efficient calculation of first order derivatives is possible with OpenModelica based on symbolic differentiation and has been successfully demonstrated using real world problems in [5]. This calculation of the derivatives benefits on the one hand from the simplification of expressions and on the other hand from the code, which is efficiently generated by OpenModelica. For optimization purposes the second order derivatives are important as well, since most of the optimization algorithms rely on them, e.g. Ipopt [19], which is used in this work. Second order derivatives are currently not symbolically available in OpenModelica, but could be provided numerically based on the already mentioned first order derivatives.

Another possibility is using the automatic differentiation tool ADOL-C, which is capable of working directly with the generated code of OpenModelica and has already been used successfully with Ipopt. Moreover, ADOL-C comes with a lot of additional features, e.g. efficient calculation of derivatives of different or-

ders. Last but not least, the implementation leads to a very fast solution with little memory costs for the underlying NOCP. The modeling and problem description is done in Modelica [8] extended with the optimization objective functions specified in Optimica [1]. This paper is organized as follows. In section 2, the mathematical representation of the nonlinear optimal control problem is discussed. The main idea of discretizing the NOCP based on orthogonal collocation principles is described in section 3. The efficient realization of the derivative calculation using ADOL-C is demonstrated in section 4. Section 5 presents the implementation details with respect to scaling, initialization and derivative calculation. Finally, the performance of the newly developed tool chain is discussed in section 6. The paper concludes the work with final remarks and suggestions for future work.

## 2 Nonlinear Optimal Control Problem (NOCP)

In many applications the NOCP is described by the following mathematical representation [10]:

$$\min_{u(t)} J(x(t), u(t), t) = E(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) dt \quad (2.1)$$

s.t.

$$x(t_0) = x_0 \quad (2.2)$$

$$\dot{x}(t) = f(x(t), u(t), t) \quad (2.3)$$

$$\hat{g}(x(t), u(t), t) \leq 0 \quad (2.4)$$

$$r(x(t_f)) = 0 \quad (2.5)$$

where  $x(t) = [x^{(1)}(t), \dots, x^{(n_x)}(t)]^\top$  and  $u(t) = [u^{(1)}(t), \dots, u^{(n_u)}(t)]^\top$  are the state vector and control variable vector for  $t \in [t_0, t_f]$ , respectively. The constraints (2.2), (2.3), (2.4) and (2.5) represent the initial conditions, the nonlinear dynamic model description based on differential algebraic equations (DAEs), the path constraints  $\hat{g}(x(t), u(t), t) \in \mathbb{R}^{n_g}$  and the terminal constraints [3]. With respect to the implementation in Ipopt and the Modelica language, it is appropriate to split the box constraints from  $\hat{g}(x(t), u(t), t) \leq 0$ , i.e.

$$\begin{array}{lclcl} x_{\min} & \leq & x(t) & \leq & x_{\max} \\ u_{\min} & \leq & u(t) & \leq & u_{\max} \end{array}$$

and to introduce so-called slack variables for the rest

$$g(x(t), u(t), t) + s(t) = 0$$

with  $s(t) \geq 0 \in \mathbb{R}^{n_g}$ . Therefore, it is possible to use the attributes `min` and `max` already available in Modelica for the description[9].

## Modelica model description

The mathematical representation of a Modelica model is typically given by DAEs

$$F(x(t), u(t), y(t), t) = 0.$$

However, most Modelica tools, especially OpenModelica, are capable of converting this formulation (by means of the so-called BLT transformation [2]) into a semi-explicit ODE form as formulated also in (2.3)

$$\begin{array}{l} \dot{x}(t) = f(x(t), u(t), t), \\ y(t) = h(x(t), u(t), t). \end{array}$$

In general, there is no closed expression for the functions  $f$  and  $g$ , but rather, iterative techniques, e.g., Newton's method, are employed to solve the so-called occurrent linear or nonlinear algebraic loops [2].

At this point it is possible to choose between two strategies for the discretization of (2.3). On the one hand, it is feasible to transform these algebraic loops into residual form and to add them subsequently to the discretized NOCP formulation. This approach has the advantage that the costs for solving the algebraic loop in each evaluation of the function  $f$  are saved. However, the solver space as well as the workload of the optimizer increases. On the other hand, the algebraic loop can be solved during each optimizer step by a linear or nonlinear solver, depending on the type of problem. This procedure might have the drawback, that the solution of these algebraic loops might be also quite time consuming, especially in the case of a highly nonlinear equation system.

This paper focuses on the second strategy. The semi-explicit ODE form is generated using OpenModelica and solved for each optimizer step. In addition, a more general NOCP formulation can be converted by means of the BLT transformation to a semi-explicit NOCP as stated in (2.1), (2.2), (2.3), (2.4) and (2.5).

## 3 Collocation

Previous work [3] has shown that solving the NOCP with a collocation approach is efficient. Therefore, the choice of collocation nodes is important, since that influences the stability and order of the integration method [7, 16]. The RADAU IIA method [4] is

one possible method to choose the collocation nodes. RADAU IIA is an implicit Runge-Kutta method and is typically described in the form [16]:

$$\begin{bmatrix} x_{i,1} - x_i \\ \vdots \\ x_{i,m} - x_i \end{bmatrix} = \Delta t_i \cdot (A \otimes I) \cdot \begin{bmatrix} f(x_{i,1}, u_{i,1}, t_{i,1}) \\ \vdots \\ f(x_{i,m}, u_{i,m}, t_{i,m}) \end{bmatrix} \quad (3.1)$$

with  $x_{i,j} = x(t_{i,j})$ ,  $x_i := x(t_i)$ ,  $u_{i,j} = u(t_{i,j})$ ,  $t_{i,j} := t_i + \tau_j \cdot \Delta t_i$  and  $t_i := t_0 + \sum_{l=1}^i \Delta t_l$  where  $\Delta t_i$ ,  $i = 0, \dots, n$  is the length of a subinterval and  $\tau_j \in [0, 1]$ ,  $j = 1, \dots, m$  are the collocation nodes.  $A$  and  $I$  are the Butcher and identity matrix, respectively. If  $\det(A) \neq 0$ , equation (3.1) can be transformed [7] to the following form

$$F_i(\cdot) := (A^{-1} \otimes I) \cdot \begin{bmatrix} x_{i,1} - x_i \\ \vdots \\ x_{i,m} - x_i \end{bmatrix} - \Delta t_i \cdot \begin{bmatrix} f(x_{i,1}, u_{i,1}, t_{i,1}) \\ \vdots \\ f(x_{i,m}, u_{i,m}, t_{i,m}) \end{bmatrix}$$

This form leads to a sparse structure for the Jacobian matrix, since in equation (3.1) the sparse structure is destroyed by multiplication of the dense matrix  $A$ .

The RADAU IIA can be interpreted as a Lagrange interpolation of the state  $x^{(l)}(t)$  [3, 4]

$$x^{(l)}(t_i + \tau \cdot \Delta t_i) \approx \sum_{j=1}^m p_j(\tau) \cdot x_{i,j}^{(l)} \text{ for } \tau \in [0, 1].$$

The corresponding Lagrangian polynomials are  $p_j$ . In order to handle constraints for  $\text{der}(u)$  the control variable  $u^{(l)}(t)$  needs to be interpreted as a Lagrange interpolation as well:

$$u^{(l)}(t_i + \tau \cdot \Delta t_i) \approx \sum_{j=1}^m p_j(\tau) \cdot u_{i,j}^{(l)}.$$

Therefore, it is possible to calculate  $\text{der}(u)$  as

$$\frac{du^{(l)}(t)}{dt} = \frac{\partial u^{(l)}(t_i + \tau \cdot \Delta t_i)}{\partial \tau} \cdot \Delta t_i \approx \Delta t_i \cdot \sum_{j=1}^m \frac{\partial p_j}{\partial \tau}(\tau) \cdot u_{i,j}^{(l)}.$$

Moreover, the constraints based on RADAU IIA result in an unbounded expression  $f(x_0, u_0, t_0)$ , since the node  $\tau_j = 0$  is not part of the RADAU IIA integration scheme, especially the expression  $u(t_0)$  is unbounded. The principle is visualized in figure 1. This issue can be addressed by using the LOBATTO IIIA method, which includes the nodes  $\tau_1 = 0$  and  $\tau_m = 1$ . Thus, the LOBATTO IIIA method yields an influence of  $u(t_0)$  on the NOCP. Therefore, the principles of the collocation discretization will be applied not only to states, but also to the control variables. This approach is referred to as total collocation [3, 18].

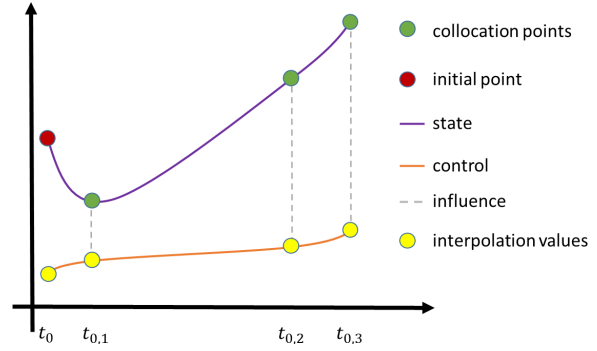


Figure 1: RADAU IIA schema

### 3.1 Lobatto IIIA

In comparison to RADAU IIA the LOBATTO IIIA method has a singular matrix  $A$  [16]. The number of nonzero elements of the Jacobian matrix can be reduced by multiplying the matrix  $A$  with  $B^{-1}$  so that

$$\begin{aligned} [0 \mid B^{-1}] \cdot A &= [0 \mid B^{-1}] \cdot \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{bmatrix} \\ &= [0 \mid B^{-1}] \cdot \begin{bmatrix} 0 & 0 \\ A_1 & B \end{bmatrix} = [\hat{A}_1 \mid I]. \end{aligned}$$

Furthermore, the equation (3.1) can be transform for the special case LOBATTO IIIA as follows

$$\begin{aligned} \hat{F}_0(\cdot) &:= (B^{-1} \otimes I) \cdot \begin{bmatrix} x_{0,2} - x_0 \\ \vdots \\ x_{0,m} - x_0 \end{bmatrix} - \\ &(A_1 \otimes I) \cdot \Delta t_0 \cdot \begin{bmatrix} f(x_{0,1}, u_{0,1}, t_{0,1}) \\ \vdots \\ f(x_{0,1}, u_{0,1}, t_{0,1}) \end{bmatrix} + \quad (3.2) \\ &\Delta t_0 \cdot \begin{bmatrix} f(x_{0,2}, u_{0,2}, t_{0,2}) \\ \vdots \\ f(x_{0,m}, u_{0,m}, t_{0,m}) \end{bmatrix} \end{aligned}$$

Note that  $x_0$  is bounded and solved with equation (2.2), so  $x_0$  is known for the optimization process. Thus, there is no need to differentiate equation (3.2) with respect to  $x_0$ , this results only in nonzero elements for  $u_0$ .

Besides, for RADAU IIA and LOBATTO IIIA applies

$$x_0 = x_{0,1} \quad \text{and} \quad x_{i+1} = x_{i,m},$$

which is solved symbolically without the optimization loop.

### 3.2 Discretized Lagrange term

Now, it is possible to use the property of the collocation method that

$$x(t_{i,j}) \approx x_{i,j}$$

for the approximation of the Lagrange term (2.1) based on quadrature formulas. Obviously, it makes sense to apply methods of Lobatto and Radau quadrature.

$$\begin{aligned} \Phi(\mathbf{x}, \mathbf{u}, \mathbf{t}) &:= \Delta t_0 \cdot \sum_{j=1}^m \hat{w}_j \cdot L_{0,j} + \sum_{i=1}^{n-1} \Delta t_i \cdot \sum_{j=1}^m w_j \cdot L_{i,j} \\ &\approx \int_{t_0}^{t_f} L(x(t), u(t), t) dt \end{aligned} \quad (3.3)$$

where  $L_{i,j} := L(x_{i,j}, u_{i,j}, t_{i,j})$ ,  $\hat{w}$  represents the Lobatto weights,  $w$  the Radau weights, and the abbreviations  $\mathbf{x} := [x_{0,1}, \dots, x_{n,m}]$ ,  $\mathbf{u} := [u_{0,1}, \dots, u_{n,m}]$ , and  $\mathbf{t} := [t_{0,1}, \dots, t_{n,m}]$ .

### 3.3 Discretized NOCP

Finally, the NOCP can be discretized:

$$\min J(\mathbf{x}, \mathbf{u}, \mathbf{s}, \mathbf{t}) = E(x_{m,n-1}, u_{m,n-1}, t_{m,n-1}) + \Phi(\mathbf{x}, \mathbf{u}, \mathbf{t})$$

s.t.

$$\begin{aligned} c(\mathbf{x}, \mathbf{u}, \mathbf{s}, \mathbf{t}) &\stackrel{!}{=} \mathbf{0} \\ u_{\max} &\leq \mathbf{u} \leq u_{\min} \\ x_{\max} &\leq \mathbf{x} \leq x_{\min} \\ \mathbf{0} &\leq \mathbf{s} \end{aligned}$$

where

$$c(\mathbf{x}, \mathbf{u}, \mathbf{s}, \mathbf{t}) := \begin{bmatrix} \hat{F}_0(\cdot) \\ g(x_{0,1}, u_{0,1}, t_{0,1}) + s_{0,1} \\ \vdots \\ g(x_{0,m}, u_{0,m}, t_{0,m}) + s_{0,m} \\ F_1(\cdot) \\ g(x_{1,1}, u_{1,1}, t_{1,1}) + s_{1,1} \\ \vdots \\ g(x_{1,m}, u_{1,m}, t_{1,m}) + s_{1,1} \\ F_n(\cdot) \\ g(x_{n,1}, u_{n,1}, t_{n,1}) + s_{n,1} \\ \vdots \\ g(x_{n,m}, u_{n,m}, t_{n,m}) + s_{n,m} \\ r(x_{n,m}, u_{n,m}, t_{n,m}) \end{bmatrix}$$

and  $x_{0,1} = x_0$  with  $\mathbf{s} = [s_{0,1}, \dots, s_{m,n}]$ ,  $s_{i,j} = s(t_{i,j})$ .

### 3.4 Nonlinear optimization

Now, the original NOCP is transformed to a nonlinear optimization problem, where the optimizer needs to find the optimal discretized control vector  $\mathbf{u}$  and to adapt  $\mathbf{x}, \mathbf{s}$  so that the constraints are fulfilled. For this operation the optimizer requires the first order derivatives from  $E(\cdot)$ ,  $\Phi(\cdot)$  and  $c(\cdot)$  as well as the second order derivatives from the Lagrangian function

$$L(z, \lambda, \mathbf{t}) = E(\cdot) + \Phi(\cdot) + \lambda^\top \cdot c(\cdot) \quad (3.4)$$

to find the solution. The sorting of  $c(\cdot)$  and  $z = [\mathbf{x}, \mathbf{u}, \mathbf{s}]$  is substantial for a good Jacobian- and Hessian-structure. The block

$$G_i(\cdot) := \begin{bmatrix} F_i(\cdot) \\ g(x_{i,1}, u_{i,1}, t_{i,1}) + s_{i,1} \\ \vdots \\ g(x_{i,m}, u_{i,m}, t_{i,m}) + s_{i,m} \end{bmatrix}$$

can be sorted more efficiently, if this is investigated in more detail. Furthermore, it applies

$$\frac{\partial x_{i,j}^{(l)}}{\partial x_{a,b}^{(c)}} = \frac{\partial x_{i,j}^{(l)}}{\partial u_{a,b}^{(d)}} = \frac{\partial u_{i,j}^{(e)}}{\partial u_{a,b}^{(d)}} = \frac{\partial u_{i,j}^{(e)}}{\partial x_{a,b}^{(c)}} = 0$$

for  $i, a = 0, \dots, n$ ,  $j, b = 1, \dots, m$ ,  $l, c = 1, \dots, n_x$ ,  $e, d = 1, \dots, n_u$  and  $l \neq c$ ,  $e \neq d$  as well as

$$\frac{\partial x_{i,j}^{(l)}}{\partial x_{i,j}^{(l)}} = 1 = \frac{\partial u_{i,j}^{(e)}}{\partial u_{i,j}^{(e)}}$$

Therefore, the Jacobian and Hessian matrices become very sparse. Furthermore, it should be taken advantage of the cyclic structure in  $c(\cdot)$ , which results from the same structure in  $G_1(\cdot), \dots, G_n(\cdot)$ .

## 4 Derivatives

There are at least three different ways to compute the derivative information required by a calculus-based optimization approach: Finite Differences, Symbolic Differentiation and Algorithmic Differentiation. The first technique, i.e., finite differences (FD), is based on the Taylor expansion and yields to relatively imprecise derivative information. Furthermore, the resulting computational cost is high in comparison to the two other approaches. For example, the gradient of a scalar-valued function with  $N$  input variables is approximated using FD with  $N + 1$  function evaluations. For these reasons, FD approximations of derivatives

will not be considered in this section. Alternatively, one may analytically derive expressions to evaluate the exact derivative based on the obtained formula. This can be done by hand, which results in an error-prone process, or automatically as provided, e.g., by Maple. Natural a Modelica compiler like OpenModelica has also capabilities to differentiate symbolically a Modelica model (see [5], [2]).

This purely symbolic method usually yields a very efficient way to compute first-order derivatives for closed-form expressions. However, the computation of higher-order derivatives or the handling of iterative solution procedures still form major challenges for this approach. The usage of algorithmic differentiation (AD), also called automatic differentiation, offers a third alternative to compute gradients, Jacobians and/or Hessians required for optimization. Based on the exploitation of the chain rule, AD provides derivative information of arbitrary order within working accuracy for a function  $F : \mathbb{R}^N \rightarrow \mathbb{R}^M$  evaluated in a code segment on a computer [14]. The complexity estimates for the two basic approaches of AD, namely the forward mode and the reverse mode, are based on the operation count  $O_F$ , i.e., the number of floating point operations required to evaluate  $y = F(z)$ . Using the forward mode, one computes the required derivatives together with the function evaluation in one sweep. This approach yields one Jacobian-vector product  $\nabla F v, v \in \mathbb{R}^N$  for no more than 2.5 times  $O_F$ . One vector-Jacobian product, or equivalently  $\nabla F^\top w, w \in \mathbb{R}^M$ , is obtained using the reverse mode in its basic form also for no more than four times  $O_F$ . If  $M = 1$ , i.e.  $\nabla F$  corresponds to the gradient of a scalar-valued function, this complexity bound for the reverse mode is completely independent of the number  $n$  of input variables. Therefore, it is also known as the *cheap gradient result*. More details about AD can be found in the books [15] and [17] as well as on the web-page [www.autodiff.org](http://www.autodiff.org).

#### 4.1 Efficient Jacobian evaluation

For the examples considered here, the Jacobian of the equality constraints  $c : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is an almost square matrix of dimension  $N$ , where  $N = (n + 1) \cdot m \cdot (n_x + n_g)$ . Hence, as a first approach to evaluate the full Jacobian, one may compute the Jacobian-vector products

$$\frac{\partial c}{\partial z_i}(z) = \nabla c(z) e_i, \quad i = 1 \dots, N,$$

where  $e_i$  denotes the  $i$ th unit vector, yielding the  $N$  rows of the Jacobian using either the symbolic approach or the forward mode of AD  $N$  times. For the forward mode of AD, the following theoretical bound of the computational cost to evaluate the full Jacobian can be shown [15]

$$\text{OPS}(\nabla c(z)) \leq 2.5NO_F,$$

where  $\text{OPS}(f)$  denotes the number of floating point operations required to evaluate  $f$ . To reduce this operation count, one may use the so-called vector forward mode, where not only one derivative information is propagated with the function evaluation but a bundle of  $p$  directional derivatives. Hence, for a so-called seed matrix  $\Sigma \in \mathbb{R}^{N \times p}$  this variant of AD yields the Jacobian-matrix product  $\nabla c(z)\Sigma \in \mathbb{R}^{N \times p}$  at a computational cost that can be bounded above by

$$\text{OPS}(\nabla c(z)\Sigma) \leq (1 + 1.5p)O_F,$$

see [15]. Using  $\Sigma = I_N$  as the identity matrix in  $\mathbb{R}^{N \times N}$ , one obtains the full Jacobian with the vector forward mode of AD at a computational cost bounded above by  $(1 + 1.5N)O_F$  instead of  $2.5NO_F$  when using the standard forward mode of AD. This makes a significant difference if  $N$  is large or the function evaluation is costly.

#### 4.2 Exploiting the structure of the Jacobian

The derivative computation described so far completely ignores any structure within the Jacobian matrix. However, for the target applications of this research project, the Jacobian of the equality constraints has a block structure as shown in Fig. 2 for the Van der Pol oscillator. When computing the full Jacobian using

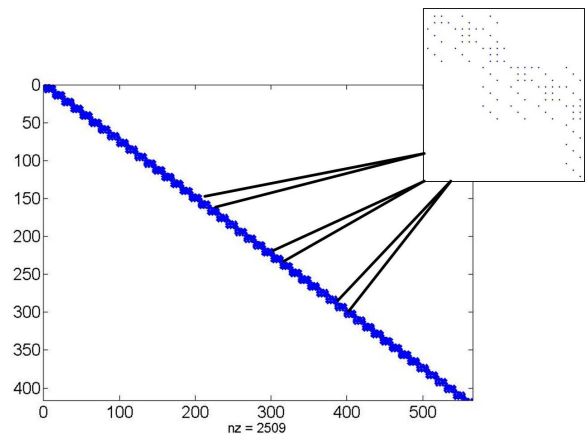


Figure 2: Van der Pol oscillator: Jacobian structure with 50 subintervals

the approach as explained in the last subsection, a lot of zero entries are computed, despite the fact that one knows that these entries are zero. To exploit the inherent structure of a sparse Jacobian, so-called compression techniques were developed. In the general case all compression techniques rely on the following four-step procedure: First determine the *sparsity structure* of the Jacobian  $\nabla c(z)$ . Second, obtain a suitable seed matrix  $\Sigma$  that defines a column partition of the Jacobian using, for example, a specialized *coloring* on the adjacency graph of the Jacobian. Then compute the *compressed* Jacobian matrix  $B = \nabla c(z)\Sigma$ . Finally, recover the numerical values of the entries of  $\nabla c(z)$  from  $B$ . The sparsity structure may be known from the application, as can be seen from the block structure in the example of the Van der Pol oscillator, or determined by a suitable variant of AD as explained in [15]. In our applications the structure is known apriori so the first two steps may be skipped and a seed matrix is available directly from the block structure. Appropriate coloring methods with the corresponding recovery strategies for the general case are discussed for example in [12]. The compressed Jacobian  $B$  is evaluated using the vector forward mode of AD. For our applications, one may consider also the sparsity structure within the blocks of the Jacobian to reduce the computational cost even further. This will be the subject of future work.

### 4.3 Exploiting the structure of the Hessian

Using a combination of the forward mode and the reverse mode of AD, one can compute Hessian-vector products for a function  $F(z)$  for a computational cost not larger than ten times  $O_F$  [15]. To exploit this facility to the full extend, for the target applications of this research project once more the sparsity of the Hessian can be taken into account. This is due to the fact that these derivative matrices have also a block structure as illustrated again for the Van der Pol oscillator in Fig. 3. The four step procedure explained above has to be adapted appropriately for the computation of second-order information in the general case. The sparsity structure of the Hessian may be known from the application, as is the case for the application discussed here, or it may be determined by a suitable variant of AD as described for example in [20]. For the general case corresponding coloring approaches together with suitable recovery strategies are presented in [13]. In our applications the seed matrix is obtained directly from the block structure known apriori.

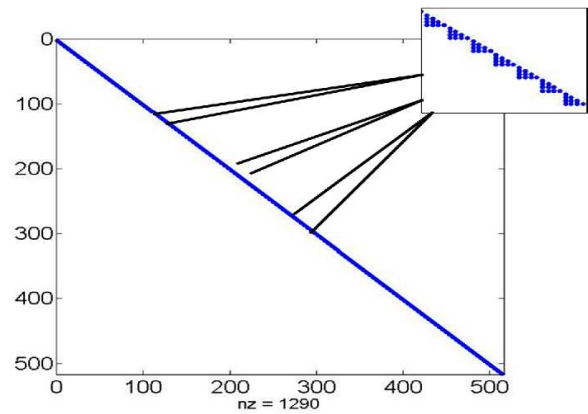


Figure 3: Van der Pol oscillator: Hessian structure with 50 subintervals

## 5 Implementation Details

The rough principle of the implementation is visualized in figure 4. At the first step the optimizer re-

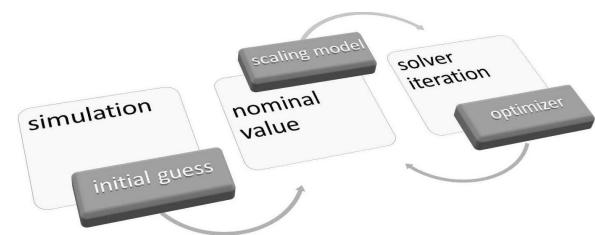


Figure 4: Implementation details

quires a sufficiently good starting point. In order to keep the constraint error for equation (2.3) small the method in OpenModelica creates a starting solution based on a simulation run. The initial guess of the control variable is set constant with the value of the start attribute. Obviously, a good setting of the initial values of the control variables can accelerate the NOCP solution process. Furthermore, this implementation supports the attribute `nominal` in Modelica for scaling variables and constraints. The effects will be presented in section 6.1.

### Coupling of OpenModelica and ADOL-C

The AD tool ADOL-C [21] uses the technique of operator overloading provided by the C++ standard to implement a wide variety of AD-based techniques. Within the research project described in this paper, the C code generation of OpenModelica was adapted such that ADOL-C can be used to evaluate the blocks in the Jacobian of the equality constraints and the blocks of the Hessian of the Lagrangian for a class of generic test problems. That is, the block structure was ex-

exploited manually. To compute the required derivative information the standard drivers `jacobian(...)` and `hessian(...)` of ADOL-C were used. For the applications considered here, the `jacobian(...)` routine uses the vector forward mode as described above. The current coupling of OpenModelica and ADOL-C allows a flexible choice between the symbolic derivative computation already implemented in OpenModelica and the AD-based derivative computation provided by ADOL-C. The structure of the coupling is illustrated in Fig. 5.

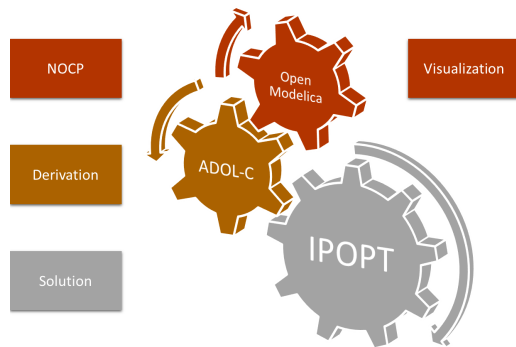


Figure 5: Structure of the coupling

## 6 Modelica Application and Performance Measurements

### 6.1 Model formulation

Currently, the user can influence the solution convergence by using native Modelica attributes like `nominal` and `start`. It should be emphasized that it is not the natural way to use `start`, since this attribute is usually reserved for initial values at start time for the simulation. Nevertheless, the initial trajectory of all problem variables is provided by a simulation, where all control variables are kept constant equal to the value of the `start` attribute. The described formulation of scaling and start trajectory will be shown on a simple model, based on the Batch Reactor found in [4]. The model was modified so that the states have the `nominal` values  $10^{10}$  and  $10^{-10}$ . The mathematical formulation is given by

$$\begin{aligned}
 & \min_{u(t)} x_2(t_f) \\
 & \text{s.t.} \\
 & x_2(t) = 10^{10} \cdot y_2(t) \\
 & x_1(t) = 10^{-10} \cdot y_1(t) \\
 & 10^{-10} \cdot \dot{y}_1(t) = - \left( u(t) + \frac{u(t)^2}{2} \right) \cdot x_1(t) \\
 & 10^{10} \cdot \dot{y}_2(t) = u(t) \cdot x_1(t) \\
 & u(t) \in [0, 5], y_1(0) = 1, y_2(0) = 0, t_f = 1
 \end{aligned}$$

which can easily be formulated in a Modelica/Optimica representation:

```

optimization BatchReactor(
    objective = cost(finalTime),
    finalTime = 1)
  Real cost = -x2;
/*DAE Modelica*/
/*states */
  Real y1(start=1e10,
          fixed=true, nominal=1e10);
  Real y2(start=0,
          fixed=true, nominal=1e-10);
/* tuner */
  input Real u(min=0, max=5, start=1.0);
protected
  Real x1;
  Real x2;
equation
  x1 = 1e-10*y1;
  x2 = 1e10*y2;
  1e-10*der(y1) = -(u+u^2/2)*x1;
  1e10*der(y2) = u*x1;
end BatchReactor;

```

When setting the correct values for the nominal attribute the solution is calculated as expected. Setting the `nominal` attribute to 1 yields the wrong result, nevertheless the optimizer finishes without any error detection.

### 6.2 Combined Cycle Power Plant

A more industry-relevant benchmark is a model of a combined cycle power plant model, see figure 6. The model contains equation-based implementations of the thermodynamic functions for water and steam, which in turn are used in the components corresponding to pipes and the boiler. The model also contains components for the economizer, the super heater, as well as the gas and steam turbines. The model has one input, 10 states, and 131 equations. For additional details on the model, see [6].

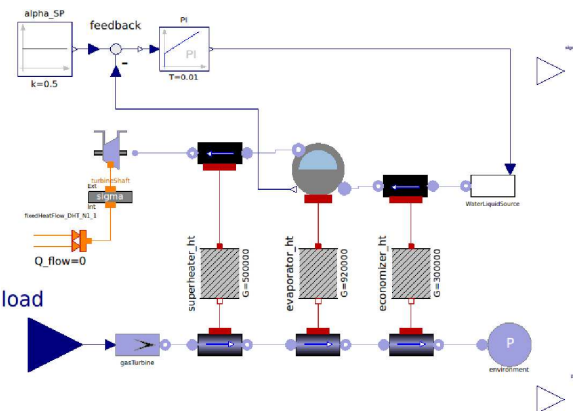


Figure 6: CombinedCycle display with OMEdit

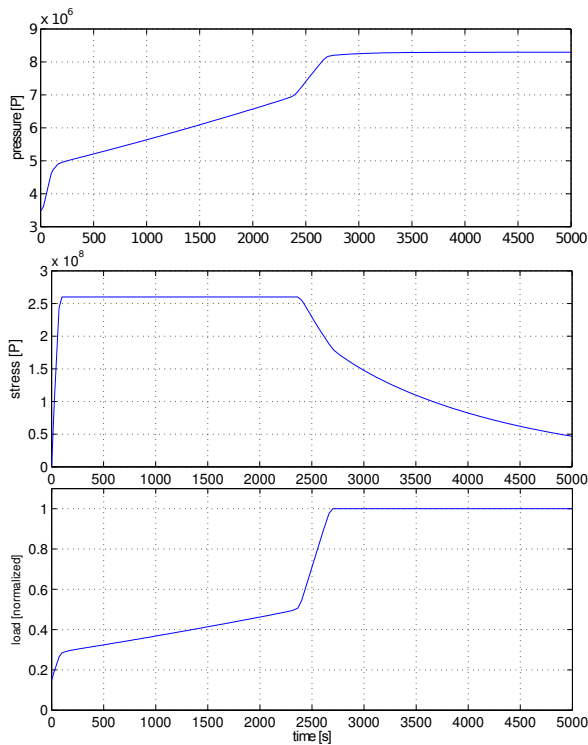


Figure 7: Optimal start-up trajectories. The upper curve shows the pressure in the evaporator, the middle curve shows the thermal stress in the steam turbine shaft and the lower curve shows the control input represented by the load.

The optimization problem is set up to use 50 collocation points that results in 1651 variables for the NOCP and was solved on a PC with a 3.2GHz Intel(R) Core(TM) i7. The algorithm requires an initial trajectory of all problem variables, which is provided by a simulation where the rate of change of the gas turbine load is set to a constant value. The optimization results are shown in figure 7 and 8 and correspond with the results that are discussed in detail in [6]. Here, the trajectories are smoother, and the performance has been improved substantially.

options		iteration	time [s]
Jacobian	Hessian		
ADOL-C	ADOL-C	39	2.29472
ADOL-C	BFGS	48	0.86425
OMC	BFGS	48	0.88558

Table 1: Time measurements of the solving process.

In table 1 the time measurements of the solving process are summarized for different options of derivatives calculation. One can see that the solution with ADOL-C needs less iterations, which is a strong indication that the solution is more accurate and more

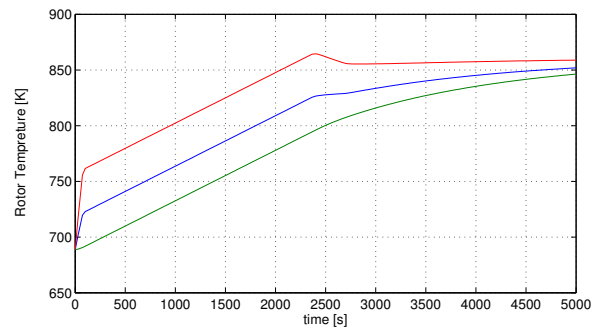


Figure 8: Optimal start-up trajectories. The upper curve shows the live steam temperature, the middle and low curves show the turbine rotor surface and mean temperatures.

stable. This is even more important for stiff models. However, the calculation of the Hessian with ADOL-C need currently a factor of three more computational time. Alternative approaches for a further improvement of the runtime needed for the Hessian calculation are the subject of current research.

## 7 Conclusions

This paper presents a newly developed tool chain for solving nonlinear optimization control problems. The underlying dynamic model formulation is done in Modelica and Optimica. The demonstrated solution method is based on orthogonal collocation methods, whereby the first interval is specially treated in order to consider control variables and their derivatives also at the initial time point. The derivative information is derived using the automatic differentiation tool ADOL-C, which efficiently calculates the corresponding Jacobian and Hessian matrices for the discretized optimization problem. Special treatments of the matrices with the focus on yielding optimal sparsity patterns with respect to block and cyclic structure are performed. The resulting optimization process proves to be stable and efficient.

## 8 Acknowledgments

This work has been partially supported by the German Ministry BMBF (BMBF Förderkennzeichen: 01IS12022I) in the ITEA2 MODRIO project. The authors also would like to acknowledge the kind support from Francesco Casella, who provided the power plant model used for benchmarks. The Open Source Modelica Consortium supports the OpenModelica work.



## References

- [1] J. Åkesson. *Languages and Tools for Optimization of Large-Scale Systems*. PhD thesis, Regler, nov 2007.
- [2] J. Åkesson, W. Braun, P. Lindholm, and B. Bachmann. Generation of sparse jacobians for the function mock-up interface 2.0. In *Proceedings of the 9th International Modelica Conference*, 2012.
- [3] B. Bachmann, L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson. Parallel multiple-shooting and collocation optimization with openmodelica. In *Proceedings of the 9th International Modelica Conference*, 2012.
- [4] L.T. Biegler. *Nonlinear programming. Concepts, algorithms, and applications in chemical processes*. SIAM, 2010.
- [5] W. Braun, S. Gallardo-Yances, B. Bachmann, and K. Link. Fast simulation of fluid models with colored jacobians. In *Proceedings of the 9th International Modelica Conference*, 2012.
- [6] F. Casella, F. Donida, and J. Åkesson. Object-oriented modeling and optimal control: A case study in power plant start-up. In *Proceedings of the 8th IFAC World Congress*, Milano, Italy, 2011.
- [7] P. Deuffhard and F.A. Bornemann. *Numerische Mathematik 2: Gewöhnliche Differentialgleichungen*. De Gruyter Lehrbuch Series. Walter De Gruyter Incorporated, 2008.
- [8] H. Elmqvist, S. E. Mattsson, and M. Otter. Modelica - a language for physical system modeling, visualization and interaction. *1999 IEEE Symposium on Computer-Aided Control System Design*, 1999.
- [9] R. Franke. Formulation of dynamic optimization problems using modelica and their efficient solution. In *Proceedings of the 2th International Modelica Conference*, pages 315–323, 2002.
- [10] T.L. Friesz. *Dynamic optimization and differential games*. Springer, 2010.
- [11] P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. Openmodelica - a free open-source environment for system modeling, simulation, and teaching. In *IEEE International Symposium on Computer-Aided Control Systems Design, 2006*, pages 1588–1595, oct. 2006.
- [12] A.H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005.
- [13] A.H. Gebremedhin, A. Tarafdar, F. Manne, and A. Pothen. New acyclic and star coloring algorithms with application to computing Hessians. *SIAM J. Sci. Comput.*, 29:1042–1072, 2007.
- [14] A. Griewank, K. Kulshreshtha, and A. Walther. On the numerical stability of algorithmic differentiation. *Computing*, 94(2-4):125–149, 2012.
- [15] A. Griewank and A. Walther. *Principles and Techniques of Algorithmic Differentiation, Second Edition*. SIAM, 2008.
- [16] E. Hairer and G. Wanner. *Solving ordinary differential equations. II: Stiff and differential-algebraic problems*. Springer, 2010.
- [17] U. Naumann. *The art of differentiating computer programs. An introduction to algorithmic differentiation*. SIAM, 2012.
- [18] A. Shitahun, V. Ruge, M. Gebremedhin, B. Bachmann, L. Eriksson, J. Andersson, M. Diehl, and P. Fritzson. Model-based dynamic optimization with openmodelica and casadi. In *Proceedings of the 7th IFAC Symposium on Advances in Automotive Control*, pages 446–451, 2013.
- [19] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.
- [20] A. Walther. Computing sparse Hessians with automatic differentiation. *ACM Trans. Math. Softw.*, 34(1), 2008. Paper 3.
- [21] A. Walther and A. Griewank. Getting started with ADOL-C. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, pages 181–202. Chapman-Hall, 2012. see also <http://www.coin-or.org/projects/ADOL-C.xml>.



# Symbolic Transformations of Dynamic Optimization Problems

Fredrik Magnusson<sup>a,\*</sup> Karl Berntorp<sup>a</sup> Björn Olofsson<sup>a</sup> Johan Åkesson<sup>a,b</sup>

<sup>a</sup>Department of Automatic Control, Lund University, SE-221 00 Lund, Sweden

<sup>b</sup>Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

## Abstract

Dynamic optimization problems involving differential-algebraic equation (DAE) systems are traditionally solved while retaining the semi-explicit or implicit form of the DAE. We instead consider symbolically transforming the DAE into an ordinary differential equation (ODE) before solving the optimization problem using a collocation method. We present a method for achieving this, which handles DAE-constrained optimization problems. The method is based on techniques commonly used in Modelica tools for simulation of DAE systems.

The method is evaluated on two industrially relevant benchmark problems. The first is about vehicle-trajectory generation and the second involves startup of power plants. The problems are solved using both the DAE formulation and the ODE formulation and the performance of the two approaches is compared. The ODE formulation is shown to have roughly three times shorter execution time. We also discuss benefits and drawbacks of the two approaches.

*Keywords: dynamic optimization, symbolic transformations, causalization, collocation*

## 1 Introduction

Industrial usage of optimization of large-scale dynamic systems has increased during the last decades. Dynamic optimization problems occur in many different fields and applications, and include parameter estimation and optimal control. Applications of optimal control include minimization of material and energy consumption during set-point transitions in power plants [1] and chemical processes [2], minimizing duration of vehicle maneuvers [3], and trajectory optimization in robotics [4].

The applications of optimal control are diverse and

occur in both online and offline settings. Online optimal control is usually done in the form of Model Predictive Control (MPC). Offline applications include finding optimal trajectories, which can be used either as a reference during manual control or as nominal trajectories combined with online feedback handling deviations due to model uncertainty and disturbances. Another offline application is the identification of system bottlenecks, for example by analyzing adjoint variables.

This paper considers models described by differential-algebraic equation (DAE) systems, and investigates the benefits of applying symbolic transformations to the DAE before applying numerical optimization methods. The DAE is transformed into an ordinary differential equation (ODE). This will often lead to a drastically reduced number of system variables, as the algebraic variables are eliminated from the equation system. On the other hand, the transformed equations will also be denser and consist of more expressions of higher complexity. This transformation is common practice in simulation of DAEs in Modelica tools [5], but is traditionally not done in the context of DAE-constrained optimization, where the DAE is instead usually retained in its natural semi-explicit or implicit form.

The technique is evaluated in two case studies. The first case concerns generation of time-optimal trajectories for road vehicles. The second case concerns optimal startup of combined-cycle power plants.

The main contributions of this paper are the demonstration of how a method commonly used in Modelica tools can be applied to dynamic optimization problems and experiments indicating the potential of the method. While similar methods have been used before [6, 7], this paper studies the properties of the approach when compared to the more traditional approach that discretizes the full DAE.

The paper outline is as follows. Section 2 presents the background of transforming low-index DAEs into ODEs by causalization, the formulation of dynamic

---

\*Corresponding author: fredrik.magnusson@control.lth.se  
The authors are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University.

optimization problems and their solution, and the tools used in the implemented framework. Section 3 discusses how the DAE causalization technique is used to transform DAE-constrained optimization problems into ODE-constrained problems. Section 4 explains the case studies used to evaluate the method and Section 5 presents the corresponding results. Finally, Section 6 concludes the paper and discusses future work.

## 2 Background

We present the standard approach of transforming a DAE into an ODE by causalization. We then discuss the formulation of general optimization problems involving dynamic systems and common methods for solving these. We finally present the tools used to implement the methods presented in this paper.

### 2.1 Causalization of DAEs

In the first step of the compilation process in a Modelica tool chain, a compiler front-end transforms Modelica source code into a flat representation, consisting essentially of lists of variables, functions, equations, and algorithms. Based on this model representation, symbolic operations such as alias elimination and index reduction are applied to reduce the size of the model and to ensure that the resulting DAE is of most index 1. In this section, we outline the steps needed to transform an implicit DAE into an ODE, which is one of the key elements of the method investigated in this paper. We introduce the following notation:

- $t$  time
- $t_f$  final time
- $x$  state (differentiated variables)
- $\dot{x}$  time derivative of state
- $y$  algebraic variables
- $u$  inputs
- $p$  parameters without predetermined values

The initial time is assumed to be 0 and the final time  $t_f$  may or may not be predetermined, but is always finite. The variables  $x, \dot{x}, y$ , and  $u$  depend on time. This dependence will be implicit in certain expressions throughout the paper.

We consider nonlinear nonhybrid index-1 DAE systems of the form

$$F(t, \dot{x}(t), x(t), y(t), u(t), p) = 0, \quad t \in [0, t_f]. \quad (1)$$

From an integrator perspective, we introduce

$$z := (\dot{x}, y), \quad v := (t, x, u, p)$$

to denote the unknown and known variables of the equation system, respectively. By reordering the arguments of  $F$ , the DAE can be written

$$F(z, v) = 0. \quad (2)$$

The conceptual idea of DAE causalization commonly used in Modelica tools is to compute the inverse relationship of  $F$ :

$$z = g(v). \quad (3)$$

The DAE can then be written as the ODE

$$\dot{x} = \bar{F}(t, x, u, p), \quad (4)$$

where the algebraic variables are internal in the right-hand side function. In general, there is no closed expression for the function  $\bar{F}$ . Rather, iterative techniques, such as Newton's method, are employed to solve algebraic loops required for computation of  $z$ .

Modelica models are typically of large scale but sparse in the sense that each model equation contains references only to a small number of equations. Graph algorithms can be employed to exploit this structure. Two commonly used algorithms that are used for this purpose are *matching algorithms*, such as the Hopcroft Karp algorithm, and Tarjan's algorithm [8] for computing *strong components*. The result of Tarjan's algorithm is used to permute the variables and equations of the DAE into Block Lower Triangular (BLT) form.

To demonstrate this procedure, let us consider an exemplary DAE system with five equations and five unknowns, where the DAE system is given by

$$\begin{aligned} F_1(z_1, z_5, v) &= 0, \\ F_2(z_3, v) &= 0, \\ F_3(z_1, z_2, z_3, z_4, v) &= 0, \\ F_4(z_1, z_3, z_5, v) &= 0, \\ F_5(z_2, z_5, v) &= 0. \end{aligned} \quad (5)$$

Note that the variable  $v = (t, x, u, p)$  is known and needs not be considered in the following analysis. The dependence on the  $z$  variables can be shown in the following incidence matrix:

	$z_1$	$z_2$	$z_3$	$z_4$	$z_5$
$F_1$	*	0	0	0	*
$F_2$	0	0	*	0	0
$F_3$	*	*	*	*	0
$F_4$	*	0	*	0	*
$F_5$	0	*	0	0	*

(6)

An asterisk in (6) in row  $i$  and column  $j$  denotes that the residual function  $F_i$  contains a reference to the variable  $z_j$ . Application of the BLT procedure yields the following permuted incidence matrix:

$$\begin{array}{c|ccccc}
 & z_3 & z_1 & z_5 & z_2 & z_4 \\
 \hline
 F_2 & * & 0 & 0 & 0 & 0 \\
 F_4 & * & * & * & 0 & 0 \\
 F_1 & 0 & * & * & 0 & 0 \\
 F_5 & 0 & 0 & * & * & 0 \\
 F_3 & * & * & 0 & * & *
 \end{array} \quad (7)$$

By reordering and grouping the variables according to

$$\bar{z}_1 := z_3,$$

$$\bar{z}_2 := (z_1, z_5),$$

$$\bar{z}_3 := z_2,$$

$$\bar{z}_4 := z_4,$$

the DAE (5) can be written

$$G_1(\bar{z}_1, v) = 0, \quad (8a)$$

$$G_2(\bar{z}_1, \bar{z}_2, v) = 0, \quad (8b)$$

$$G_3(\bar{z}_2, \bar{z}_3, v) = 0, \quad (8c)$$

$$G_4(\bar{z}_1, \bar{z}_2, \bar{z}_3, \bar{z}_4, v) = 0, \quad (8d)$$

where the functions  $G_i$  are constructed from the functions  $F_j$ . For demonstrative purposes, we assume that (8a) and (8d) can be solved explicitly for  $\bar{z}_1$  and  $\bar{z}_2$  respectively, and that (8b) and (8c) can not be solved analogously. The DAE can then be represented by the following sequence of assignment statements and implicit equation systems:

$$\bar{z}_1 \leftarrow g_1(v), \quad (9a)$$

$$G_2(\bar{z}_1, \bar{z}_2, v) = 0, \quad (9b)$$

$$G_3(\bar{z}_2, \bar{z}_3, v) = 0, \quad (9c)$$

$$\bar{z}_4 \leftarrow g_4(\bar{z}_1, \bar{z}_2, \bar{z}_3, v). \quad (9d)$$

where (9b) and (9c) require iterative methods to be solved. It is typical for Modelica models to contain only a small number of implicit equation systems that require iteration and a large number of trivial, for example linear, equations that can be solved symbolically.

For a general DAE, the BLT procedure results in a sequence of equation systems of the form

$$\begin{array}{l}
 \bar{G}_1(\bar{z}_1, v) = 0, \\
 \vdots \\
 \bar{G}_i(\bar{z}_1, \dots, \bar{z}_i, v) = 0, \\
 \vdots \\
 \bar{G}_b(\bar{z}_1, \dots, \bar{z}_b, v) = 0,
 \end{array} \quad (10)$$

where  $b$  is the number of blocks in the BLT form and the unknown of each equation is  $\bar{z}_i$ . Some equations can be solved explicitly by symbolic manipulation, while the rest needs to be solved iteratively.

Given values of the known variables in  $v$ , the sequence of solved and unsolved blocks (10) allows for the computation of the corresponding state derivative and algebraic vectors contained in  $z$ . Accordingly, the DAE has been causalized into an ODE of the form (4).

We consider the class of DAEs that can be transformed into an ODE where no implicit systems of equations need to be solved. This lets us redefine each unknown  $\bar{z}_i$  to be a single scalar variable and compute it explicitly as

$$\bar{z}_i = g_i(\bar{z}_1, \dots, \bar{z}_{i-1}, v), \quad i = 1, \dots, n_z, \quad (11)$$

where  $n_z$  is the total number of states and algebraic variables. While this class of systems is limited, the proposed method is trivially extendible to systems containing implicit systems by simply exposing the implicit systems to the numerical optimization method used in the end, in which case some algebraic equations will remain in the transformed DAE.

## 2.2 Dynamic optimization

Consider the DAE-constrained optimization problem:

$$\begin{aligned}
 &\text{minimize } \phi(t_f, \dot{x}(t_f), x(t_f), y(t_f), u(t_f), p) \\
 &\quad + \int_0^{t_f} L(t, \dot{x}(t), x(t), y(t), u(t), p) dt, \quad (12a)
 \end{aligned}$$

$$\text{w.r.t. } t_f, \dot{x}, x, y, u, p,$$

$$\text{subject to } F(t, \dot{x}, x, y, u, p) = 0, \quad (12b)$$

$$F_0(\dot{x}(0), x(0), y(0), p) = 0, \quad (12c)$$

$$\dot{x}_L \leq \dot{x}(t) \leq \dot{x}_U, \quad (12d)$$

$$x_L \leq x(t) \leq x_U, \quad (12e)$$

$$y_L \leq y(t) \leq y_U, \quad (12f)$$

$$u_L \leq u(t) \leq u_U, \quad (12g)$$

$$p_L \leq p \leq p_U, \quad (12h)$$

$$h_e(t, \dot{x}, x, y, u, p) = 0, \quad (12i)$$

$$h_i(t, \dot{x}, x, y, u, p) \leq 0, \quad (12j)$$

$$H_e(\dot{x}(t_f), x(t_f), y(t_f), u(t_f), p) = 0, \quad (12k)$$

$$H_i(\dot{x}(t_f), x(t_f), y(t_f), u(t_f), p) \leq 0, \quad (12l)$$

$$\forall t \in [0, t_f].$$

The objective (12a) consists of the terminal cost  $\phi$  and the integral of  $L$ , which is the accumulated cost. Constraint (12b) is the DAE describing the system dynam-

ics. Constraint (12c) enforces the DAE initial conditions, which are often given on the form  $x(0) = x_0$ . Constraints (12d)–(12h) are variable bounds. Constraints (12i) and (12j) are path constraints on equality and inequality form, which can be seen as generalizations of the variable bounds, where the functions  $h_e$  and  $h_i$  define the boundary. The variable bounds are separated from the path constraints because some solvers allow for more efficient treatment of the bounds. Constraints (12k) and (12l) are terminal constraints on equality and inequality form. These are similar to the path constraints, but instead of being enforced at all points in time they are only enforced at  $t_f$ .

There are many approaches to solving dynamic optimization problems of the form (12). Until the 1970s, problems were solved using dynamic programming or Pontryagin’s maximum principle. These approaches are ill-suited for large-scale problems and problems with inequality constraints. Modern techniques often involve finding an approximate solution to the infinite-dimensional optimization problem by transcribing it into a finite-dimensional nonlinear program (NLP). These are called direct methods. The main difference among direct methods is how to handle the dynamic equations of the system. This paper employs direct local collocation. Another common approach is direct multiple shooting. See [9, 10] for overviews on direct local collocation and other direct methods.

The main idea of direct local collocation is to first divide the time horizon into a certain number of elements. Then within each element, the constraints (12b) and (12d)–(12j) are enforced at only a finite number of points, called collocation points, instead of at every point in the element. There are different schemes for choosing the placement of the collocation points with different numerical properties. The results in Section 5 have been generated using Radau points.

The constraints resulting from the collocation procedure are considered as interpolation conditions on the time-dependent variables  $x$ ,  $y$ , and  $u$ . Thus the sought approximate optimal trajectories to (12) become piecewise-polynomial, where the degrees of the polynomials are determined by the number of collocation points. The state derivative  $\dot{x}$  is obtained by differentiating the corresponding polynomials for the state  $x$ . The integral term in the objective (12a) is approximated as a sum using quadrature. See [11] for a complete description of the used collocation method, and also possible generalizations of (12). Once the discretization procedure is completed, the infinite-

dimensional dynamic optimization problem (12) has been transformed into an NLP of the following general form:

$$\text{minimize} \quad \tilde{f}(\tilde{x}), \tag{13a}$$

$$\text{with respect to} \quad \tilde{x} \in \mathbb{R}^{n_{\tilde{x}}},$$

$$\text{subject to} \quad \tilde{x}_L \leq \tilde{x} \leq \tilde{x}_U, \tag{13b}$$

$$\tilde{g}(\tilde{x}) = 0, \tag{13c}$$

$$\tilde{h}(\tilde{x}) \leq 0. \tag{13d}$$

The number of discretization points is affected both by the number of elements and the number of collocation points within each element. An increase in either of these directly corresponds to an increase in the number of variables and constraints in the NLP (13).

In this work, (13) is solved using a gradient-based method. This requires the NLP functions  $f$ ,  $g$ , and  $h$  to be twice continuously differentiable with respect to all of the NLP variables  $\tilde{x}$ . In particular, this requirement implies differentiability of the DAE-residual  $F$ , which excludes the possibility of solving optimization problems involving hybrid systems.

When using direct methods for dynamic optimization problems involving DAEs, the time discretization method is typically applied to the DAE in its natural semi-explicit or implicit form [9, 10]. In this paper, we instead consider causalizing the DAE as described in Section 2.1 and then eliminating the algebraic variables in the optimization problem as described in Section 3. The result is an ODE-constrained optimization problem of the following form:

$$\begin{aligned} \text{minimize} \quad & \bar{\phi}(t_f, \dot{x}(t_f), x(t_f), u(t_f), p) \\ & + \int_0^{t_f} \bar{L}(t, \dot{x}(t), x(t), u(t), p) dt, \end{aligned} \tag{14a}$$

$$\text{with respect to} \quad t_f, \dot{x}, x, u, p,$$

$$\text{subject to} \quad \dot{x} = \bar{F}(t, x, u, p) = 0, \tag{14b}$$

$$\bar{F}_0(\dot{x}(0), x(0), p) = 0, \tag{14c}$$

$$\dot{x}_L \leq \dot{x}(t) \leq \dot{x}_U, \tag{14d}$$

$$x_L \leq x(t) \leq x_U, \tag{14e}$$

$$u_L \leq u(t) \leq u_U, \tag{14f}$$

$$p_L \leq p \leq p_U, \tag{14g}$$

$$\bar{h}_e(t, \dot{x}, x, u, p) = 0, \tag{14h}$$

$$\bar{h}_i(t, \dot{x}, x, u, p) \leq 0, \tag{14i}$$

$$\bar{H}_e(\dot{x}(t_f), x(t_f), u(t_f), p) = 0, \tag{14j}$$

$$\bar{H}_i(\dot{x}(t_f), x(t_f), u(t_f), p) \leq 0, \tag{14k}$$

$$\forall t \in [0, t_f].$$

The objective and constraint functions occurring in

problem (14) are defined analogously to those occurring in problem (12).

This paper investigates the possibilities of transforming problems on the form of (12) to the form of (14) and the impact this has on the solution of the NLPs resulting from collocation methods.

### 2.3 Tools

The proposed method has been implemented using the open-source platform JModelica.org [12]. JModelica.org is a tool targeting simulation and optimization of large-scale dynamic systems. The systems are described using Modelica, and the optimization is formulated with the Modelica extension Optimica [13].

The framework uses IPOPT [14] to solve the NLP (13). IPOPT uses a sparse primal-dual interior point method to find local optima to large-scale NLPs. To this end, it uses first- and second-order derivatives of the NLP functions  $f$ ,  $g$ , and  $h$  in problem (13). The implemented framework uses CasADi [15] (Computer algebra system with Automatic Differentiation) to obtain these derivatives, and also to perform the transformation from (12) to (14). CasADi is a low-level tool for efficiently computing derivatives using algorithmic differentiation (AD) while preserving sparsity, and is tailored for dynamic optimization.

## 3 Proposed method

This section presents the proposed method for transforming the DAE-constrained optimization problem (12) into an ODE-constrained optimization problem of the form (14). We then compare the properties of the untransformed problem to those of the transformed problem.

### 3.1 Problem transformation

The first step is the causalization of the DAE in (12b), as described in Section 2.1. Under the assumption that all equations in (10) can be solved explicitly by symbolic manipulations, this yields the system of equations (compare with (11))

$$\bar{z}_i = g_i(\bar{z}_1, \bar{z}_2, \dots, \bar{z}_{i-1}, v), \quad i = 1, \dots, n_z, \quad (15)$$

where  $\bar{z}_i$  is a state derivative or an algebraic variable. The explicit solution for  $\bar{z}_1, \bar{z}_2, \dots, \bar{z}_{i-1}$  is then inlined into (15). The resulting equations are described by the

following recursive relations:

$$\bar{g}_1(v) := g_1(v), \quad (16a)$$

$$\bar{g}_i(v) := g_i(\bar{g}_1(v), \bar{g}_2(v), \dots, \bar{g}_{i-1}(v), v), \quad (16b)$$

$$\bar{z}_i = \bar{g}_i(v), \quad i = 1, \dots, n_z. \quad (16c)$$

By expanding the variables  $v$  and  $z$  and separating the state derivatives from the algebraic variables in (16c), the equations can be written in the form

$$\dot{x} = \bar{F}(t, x, u, p), \quad (17a)$$

$$y = k(t, x, u, p), \quad (17b)$$

where each scalar component of  $\bar{F}$  and  $k$  is equal to  $\bar{g}_i$  for some  $i$ . Thus (17a) gives rise to the constraint in (14b). Equation (17b) is used to eliminate the algebraic variables in the objective function and remaining constraints of (12). To demonstrate, the function  $h_i$  in (12j) is transformed according to

$$\begin{aligned} h_i(t, \dot{x}(t), x(t), y(t), u(t), p) \\ &= h_i(t, \dot{x}(t), x(t), k(t, x(t), u(t), p), u(t), p) \\ &=: \bar{h}_i(t, \dot{x}(t), x(t), u(t), p). \end{aligned}$$

Constraint (12f) is not possible to transform in this manner, since the algebraic variables have been eliminated. This is handled by transforming (12f) into its more general form (12j), which has consequences discussed in Section 3.2, and then transforming it as demonstrated above. By eliminating the algebraic variables in the objective and remaining constraints in the same manner, the optimization problem (12) has been transformed into the equivalent problem (14).

### 3.2 Method properties

A significant benefit of transforming the original DAE-constrained problem into an ODE-constrained problem is the reduction of system variables and equations, which directly leads to a smaller NLP after applying a collocation method. However, while the constraints will be fewer in number their expressions will be more complex, which may lead to expression graphs that in the end require more memory to store and more time to evaluate than the expression graphs in the original problem. On the other hand, component-based physical modeling will often lead to a great amount of trivial algebraic equations that can be solved explicitly without increasing expression complexity. It thus stands to reason that the proposed method is well suited to be used in a Modelica framework.

The elimination of algebraic variables will make the incidence matrix for the dynamic equation system

denser. However, the sparsity in the NLP does not mainly stem from the structure of the dynamic equations, but rather from the largely decoupled dependency of NLP variables representing system variable values in different elements. Hence in many cases, the loss of sparsity in the DAE will not have a significant impact on the solution of the NLP constructed by collocation methods.

When employing interior-point methods to solve NLPs originating from dynamic optimization problems, it is often beneficial to introduce artificial bounds on system variables to prevent the solver from leaving the domains of the involved functions. When the algebraic variables are eliminated from the problem, it is no longer possible to use these artificial bounds on algebraic variables. This can lead to difficulties in obtaining convergence in the numerical solver.

Experience has shown that seemingly trivial modifications in the original problem formulation can lead to drastically different convergence behavior in the NLP solver. By explicitly solving these trivial equations by performing the proposed transformation, increased robustness to small model modifications is attained.

The inlining step performed to obtain (16c) gives rise to a great amount of expression duplication in the expression graph for the right-hand side of (16b) because of the recursive nature of the equations. This can be avoided by storing the expressions used to evaluate  $\bar{g}_i$  in a function and calling these functions when evaluating  $g_i$ . This paper does, however, not consider this proposition any further.

## 4 Benchmark problems

Two different optimal-control cases are presented for evaluation of the proposed transformation method: vehicle trajectory generation and startup of a combined-cycle power plant. The vehicle models are implemented in a flat manner, with condensed and complex equations, whereas the power plant model is implemented in an object-oriented fashion with a large number of simple equations.

### 4.1 Vehicle trajectory generation

The first considered case is a minimum-time problem, where we seek the time-optimal maneuver for an automobile in a hairpin turn, see Figure 1. A methodology for solving this kind of problems has previously been investigated in [3, 16, 17]. We use two different chassis models in the evaluation. The first is a single-track model [18], where the two wheels on each axle

are lumped together. The model has three degrees of freedom: two translational and one rotational.



Figure 1: An example of a hairpin turn. Photo courtesy of RallySportLive.

The second model is a double-track model [19] with five degrees of freedom: two translational and three rotational. The suspension dynamics model is a rotational spring-damper system, and longitudinal and lateral load transfer is included.

The lateral slip  $\alpha$  and the longitudinal slip  $\lambda$  are defined as in [20]:

$$\dot{\alpha}_i \frac{\sigma}{v_{x,i}} + \alpha_i := -\arctan\left(\frac{v_{y,i}}{v_{x,i}}\right), \quad (18)$$

$$\lambda_i := \frac{r\omega_i - v_{x,i}}{v_{x,i}}, \quad (19)$$

where  $\sigma$  is the relaxation length,  $r$  is the wheel radius,  $\omega_i$  is the wheel angular velocity for wheel  $i$ , and  $v_{y,i}$  and  $v_{x,i}$  are the lateral and longitudinal wheel velocities for wheel  $i$  with respect to an inertial system, expressed in the coordinate system of the wheel.

The tire forces  $F_{x0}$  and  $F_{y0}$  for the longitudinal and lateral directions under pure slip conditions are computed with the Magic formula [20], given by

$$F_{x0,i} = \mu_x F_{z,i} \sin\left(C_{x,i} \arctan\left(B_{x,i} \lambda_i - E_{x,i}(B_{x,i} \lambda_i - \arctan B_{x,i} \lambda_i)\right)\right), \quad (20)$$

$$F_{y0,i} = \mu_y F_{z,i} \sin\left(C_{y,i} \arctan\left(B_{y,i} \alpha_i - E_{y,i}(B_{y,i} \alpha_i - \arctan B_{y,i} \alpha_i)\right)\right), \quad (21)$$

for each wheel  $i = 1, \dots, 4$ . In (20)–(21),  $\mu_x$  and  $\mu_y$  are friction coefficients and  $B$ ,  $C$ , and  $E$  are parameters.

We have chosen two different approaches for modeling the tire forces under combined slip constraint, both of which are described next. A straightforward model of combined slip is based on the friction ellipse, and is described by the elliptical relation

$$F_{y,i} = F_{y0,i} \sqrt{1 - \left(\frac{F_{x0,i}}{\mu_{x,i} F_{z,i}}\right)^2}, \quad (22)$$



where  $F_{x0}$  is used as an input variable [21]. However, here the driving/braking torques are used as input. The main limitation with this model is that the longitudinal force does not explicitly depend on the lateral slip.

Another approach to tire modeling, which is inspired by the Magic Formula, is to scale the nominal forces (20)–(21) with weighting functions  $G_{\alpha,i}$  and  $G_{\lambda,i}$ , which depend on  $\alpha$  and  $\lambda$  [20]. The relations in the longitudinal direction are

$$H_{\alpha,i} = B_{1,i} \cos(\arctan(B_{2,i}\lambda_i)), \quad (23)$$

$$G_{\alpha,i} = \cos(C_{\alpha,i} \arctan(H_{\alpha,i}\alpha_i)), \quad (24)$$

$$F_{x,i} = F_{x0,i} G_{\alpha,i}, \quad (25)$$

and the corresponding relations in the lateral direction are given by

$$H_{\lambda,i} = B_{1,i} \cos(\arctan(B_{2,i}\alpha_i)), \quad (26)$$

$$G_{\lambda,i} = \cos(C_{\lambda,i} \arctan(H_{\lambda,i}\lambda_i)), \quad (27)$$

$$F_{y,i} = F_{y0,i} G_{\lambda,i}. \quad (28)$$

To summarize, four different model configurations were investigated for the vehicle trajectory generation, all having three control inputs. The combination of single-track chassis model and friction ellipse for tire modeling (ST-FE) has 13 states and 13 algebraic variables. The corresponding numbers for the combination of single-track chassis model and weighting functions (ST-WF) are 13 states and 23 algebraic variables. Considering the double-track model with friction ellipse model (DT-FE), it has 21 states and 36 algebraic variables. For the combination of double-track model and weighting functions (DT-WF), the corresponding numbers are 21 states and 56 algebraic variables.

For each model configuration, the time-optimal trajectories in the hairpin turn are to be determined. An initialization procedure based on a driver model presented in [18] is used. The optimization problem is formulated over the time horizon  $t \in [0, t_f]$  and the objective of the optimization is to minimize the final time  $t_f$  of the maneuver. Accordingly, the dynamic optimization problem to be solved can be written as:

$$\begin{aligned} & \text{minimize} && t_f, \\ & \text{subject to} && T_{i,\min} \leq T_i \leq T_{i,\max}, \quad i \in \{1, 2, 3, 4\}, \\ & && |\dot{T}_i| \leq \dot{T}_{i,\max}, \quad i \in \{f, r\}, \\ & && |\delta| \leq \delta_{\max}, \quad |\dot{\delta}| \leq \dot{\delta}_{\max} \\ & && x(0) = x_0, \\ & && x(t_f) = x_{t_f}, \quad y(t_f) = y_{t_f}, \\ & && \Gamma(X_p, Y_p) \leq 0, \\ & && F(\dot{x}, x, y, u) = 0, \end{aligned} \quad (29)$$

where  $x_0$  are the initial conditions for the state variables,  $x_{t_f}$  and  $y_{t_f}$  are the desired values at the final time  $t = t_f$ , and  $(X_p, Y_p)$  is the position of the center-of-gravity of the vehicle. The wheel driving and braking torques  $T = (T_f \ T_r)$  of the front and rear wheel axles, as well as the steer angle  $\delta$  of the front wheels are considered as inputs. The inputs are equally distributed between the wheels at the respective axle, that is,  $T_1 = T_2 = T_f/2$  and  $T_3 = T_4 = T_r/2$  for the double-track chassis model. In practice, the terminal constraints are only applied to a subset of the model variables. Further,  $\Gamma(X_p, Y_p)$  is a mathematical description of the road constraint for the center-of-gravity of the vehicle for the maneuver. These constraints in the geometric two-dimensional  $XY$ -plane are formulated as super-ellipses.

Figure 2 shows the geometric path and the time-optimal control inputs obtained for ST-WF. For more details about the solution method and the model parameters used, see [3, 17].

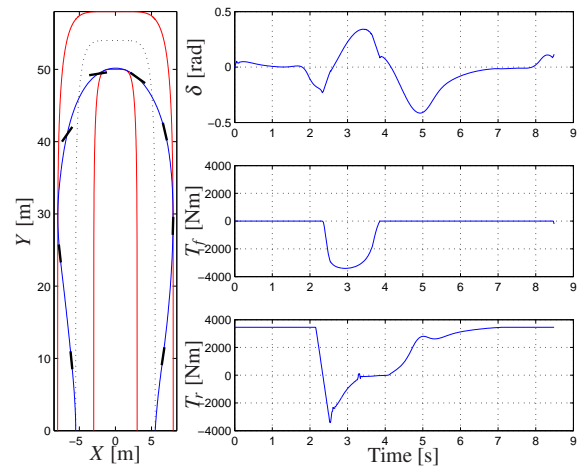


Figure 2: The geometric path and control inputs are shown for the time-optimal hairpin maneuver when using a single-track model with the weighting functions. The black bars in the left plot indicate the direction of the car every second.

## 4.2 Combined-cycle power plant startup

The second considered case concerns optimal startup of combined-cycle power plants (CCPP). The model used is described in [1]. The model has 9 states, 128 algebraic variables, and 1 control variable. The task is to minimize the time required to perform a warm startup of the power plant. This problem has become highly industrially relevant during the last years, due to an increasing need to improve power-generation flexibility. The startup process is considered finished when

the normalized load input signal  $u$  to the steam turbine, starting at 15 %, has reached 100 % and the evaporator pressure  $p$ , which is a state with an initial value of approximately 3.47 MPa, has reached 8.35 MPa.

In order to reduce the wear and tear on the steam turbine, which is one of the most expensive parts of the power plant, the thermal stress in the turbine  $\sigma$ , which is an algebraic variable, may not exceed 260 MPa. This is the main limiting factor in the startup process. Another imposed constraint is that the derivative of the load input signal  $u$  may not be negative and may not exceed  $0.1/60 \text{ s}^{-1}$ . Since these bounds are applied to the derivative of the control variable, which is not supported by the current framework, we introduce the control variable  $\dot{u}$  and add the equation

$$\frac{du}{dt} = \dot{u}.$$

This converts the control variable  $u$  into a state, giving us a total of 10 states, and the control variable is now instead  $\dot{u}$ , on which we can impose the discussed bounds. The model diagram is displayed in Figure 3.

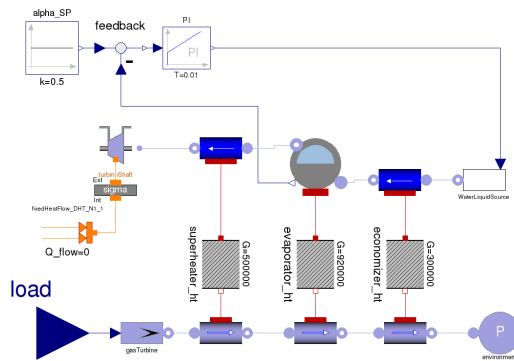


Figure 3: Power plant model diagram

The objective function is the weighted square deviation of the load input signal and the evaporator pressure from their respectively desired values, given by

$$f(z) = \int_0^{t_f} \left( 10^{-12} \cdot (p(t) - 8.35 \cdot 10^6)^2 + 0.5 \cdot (u(t) - 1)^2 \right) dt.$$

The final time is chosen to be  $t_f = 4000 \text{ s}$ . The obtained solution is displayed in Figure 4.

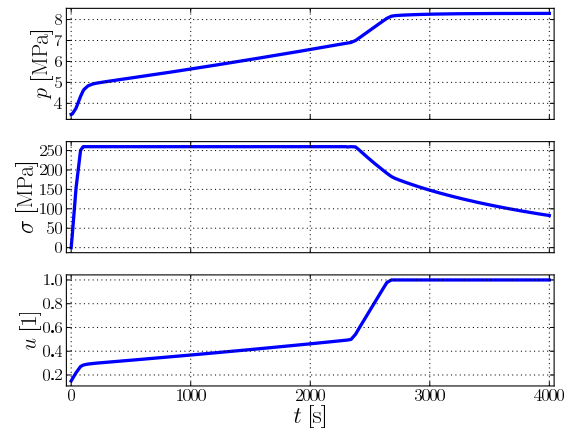


Figure 4: Optimal power plant startup

## 5 Results

The problems in the respective cases described in Section 4 were solved with the implementation described in Section 2.3. The solutions were obtained using JModelica.org revision [5625] and IPOPT version 3.11.3 with the linear solver MA57 [22]. The collocation discretization was done using 150 elements with 3 collocation points for each vehicle maneuver problem and 40 elements with 4 collocation points for the power plant startup problem. The solution procedure consists of the following three steps:

1. Model compilation, where the compiler of JModelica.org generates XML code that describes the system equations and optimization formulation. This code is then parsed by CasADi, which then creates symbolic representations of all the problem expressions.
2. Offline NLP setup, where the symbolic expressions created by CasADi in the previous step are used to construct the corresponding NLP by collocation. First- and second-order derivatives are computed by algorithmic differentiation while preserving sparsity.
3. Online NLP solution, where IPOPT solves the NLP constructed in the previous step. This is the only part of the solution procedure that would need to be performed in an online setting, such as MPC.

All problems were solved both with the model dynamics on DAE form and transformed to ODE form.

If the DAE is transformed into an ODE, the transformation takes place in the first step and is performed by CasADi. The respectively obtained solutions for the DAE and ODE formulations are the same up to tolerances.

For comparison of the two different strategies to solving the optimization problems, the time spent in step 2 and 3 of the solution procedure were measured separately for each of the optimization runs. The number of iterations required by IPOPT and the number of NLP variables have also been recorded. Table 1 displays the resulting numbers, where times are presented in seconds. The time spent in step 1 of the solution procedure is between 1 and 2 seconds for all of the problems and is largely unaffected by whether the ODE transformation is performed. These times are thus not presented.

Table 1: Solution times [s] for the considered model configurations with DAE and ODE form in the optimal control problem, respectively. In addition, the number of iterations required to solve the NLP and the total number of NLP variables are shown.

Problem		Offline	Online	Iter	NLP
ST-FE	DAE	3.6	10.6	112	20880
	ODE	3.5	5.0	83	15017
ST-WF	DAE	4.2	17.6	102	25390
	ODE	3.7	5.1	77	15017
DT-FE	DAE	9.0	152.2	303	39661
	ODE	10.5	46.0	151	23425
DT-WF	DAE	9.1	229.6	364	48681
	ODE	10.8	116.4	322	23425
CCPP	DAE	3.5	5.4	109	23574
	ODE	1.8	1.4	79	3771

There is no clear trend for the offline execution time in the four vehicle problems, whereas it is halved for the power plant. Both the number of iterations and the online execution times are shorter for the transformed problem in all compared scenarios. For the power plant, the difference in online execution time is approximately a factor of 4, whereas the vehicle examples exhibit a factor of between 2 and 3. This indicates that models containing a large amount of simple equations gain more from the proposed method, for reasons discussed in Section 3.2, but also that models with mainly complex algebraic equations gain speedups in the online NLP solution from the transformation.

## 6 Conclusions

We have presented a method for symbolically transforming a broad class of DAE-constrained optimization problem into an ODE-constrained optimization problem. The approach has been evaluated by measuring execution times for benchmark problems involving time-optimal trajectory generation for vehicles and startup of combined-cycle power plants.

The considered problems have been solved with both a traditional approach where the DAE is left intact in an implicit form, and with the presented method where the DAE system is symbolically transformed into an ODE before applying discretization techniques. Significant speedups have been observed in the solution of the NLPs resulting from a direct collocation discretization method.

While the method has exhibited great performance improvements, potential drawbacks have also been discussed. The most significant drawback is that while the dynamic equation system becomes smaller in size, the resulting expressions are often of much higher complexity.

Future work is the resolution of some of the discussed drawbacks of the approach, by only eliminating a suitable subset of the algebraic variables. A possible improvement in the opposite direction is to employ tearing techniques, which are often used for simulation of DAEs, to further reduce the number of variables exposed to the collocation method and the NLP solver. The interaction between the proposed method and other types of NLP solution methods, such as active set methods, is also worth investigating.

## Acknowledgments

We acknowledge Francesco Casella from Politecnico di Milano for granting the usage of the combined-cycle power plant model.

## References

- [1] F. Casella, F. Donida, and J. Åkesson, "Object-oriented modeling and optimal control: A case study in power plant start-up," in *18th IFAC World Congress*, (Milano, Italy), Aug. 2011.
- [2] P.-O. Larsson, J. Åkesson, and N. Andersson, "Economic cost function design and grade change optimization for a gas phase polyethylene reactor," in *50th IEEE Conf. Decision and*

- Control and European Control Conference*, (Orlando, FL), Dec. 2011.
- [3] K. Berntorp, B. Olofsson, K. Lundahl, B. Bernhardsson, and L. Nielsen, “Models and methodology for optimal vehicle maneuvers applied to a hairpin turn,” in *Am. Control Conf. (ACC)*, (Washington, DC), pp. 2139–2146, 2013.
- [4] B. Olofsson, H. Nilsson, A. Robertsson, and J. Åkesson, “Optimal tracking and identification of paths for industrial robots,” in *18th IFAC World Congress*, (Milano, Italy), Aug. 2011.
- [5] F. E. Cellier and E. Kofman, *Continuous System Simulation*. New York, NY: Springer, Mar. 2006.
- [6] B. Bachmann, L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson, “Parallel multiple-shooting and collocation optimization with OpenModelica,” in *9th Int. Modelica Conf.*, (Munich, Germany), Sept. 2012.
- [7] R. Franke, “Formulation of dynamic optimization problems using Modelica and their efficient solution,” in *2nd Int. Modelica Conf.*, (Oberpfaffenhofen, Germany), Mar. 2002.
- [8] R. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM J. Computing*, vol. 1, no. 2, pp. 146–160, 1972.
- [9] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, Philadelphia, PA: Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.
- [10] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming: Second Edition*. Advances in Design and Control, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2010.
- [11] F. Magnusson and J. Åkesson, “Collocation methods for optimization in a Modelica environment,” in *9th Int. Modelica Conf.*, (Munich, Germany), Sept. 2012.
- [12] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, “Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem,” *Computers and Chemical Engineering*, vol. 34, pp. 1737–1749, Nov. 2010.
- [13] J. Åkesson, “Optimica—an extension of Modelica supporting dynamic optimization,” in *6th Int. Modelica Conf.*, (Bielefeld, Germany), Mar. 2008.
- [14] A. Wächter and L. T. Biegler, “On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [15] J. Andersson, J. Åkesson, and M. Diehl, “CasADi – A symbolic package for automatic differentiation and optimal control,” in *Recent Advances in Algorithmic Differentiation*, Lecture Notes in Computational Science and Engineering, Berlin, Germany: Springer, 2012.
- [16] B. Olofsson, K. Lundahl, K. Berntorp, and L. Nielsen, “An investigation of optimal vehicle maneuvers for different road conditions,” in *7th IFAC Symp. Advances in Automotive Control (AAC)*, (Tokyo, Japan), pp. 66–71, 2013.
- [17] K. Lundahl, K. Berntorp, B. Olofsson, J. Åslund, and L. Nielsen, “Studying the influence of roll and pitch dynamics in optimal road-vehicle maneuvers,” in *23rd Int. Symp. Dynamics of Vehicles on Roads and Tracks (IAVSD)*, (Qingdao, China), 2013.
- [18] R. Rajamani, *Vehicle Dynamics and Control*. Berlin Heidelberg: Springer-Verlag, 2006.
- [19] K. Berntorp, “Derivation of a six degrees-of-freedom ground-vehicle model for automotive applications,” Technical Report ISRN LUTFD2/TFRT--7627--SE, Department of Automatic Control, Lund University, Sweden, Feb. 2013.
- [20] H. B. Pacejka, *Tire and Vehicle Dynamics*. Oxford, United Kingdom: Butterworth-Heinemann, 2nd edition ed., 2006.
- [21] J. Wong, *Theory of Ground Vehicles*. Hoboken, NJ: John Wiley & Sons, 2008.
- [22] HSL, “A collection of fortran codes for large scale scientific computation.” <http://www.hs1.rl.ac.uk>, 2013.

# Modelling a Lignite Power Plant in Modelica to Evaluate the Effects of Dynamic Operation and Offering Grid Services

M. Hübel<sup>2</sup>, A. Berndt<sup>2</sup>, S. Meinke<sup>1</sup>, M. Richter<sup>2</sup>, P. Mutschler<sup>2</sup>,  
E. Hassel<sup>2</sup>, H. Weber<sup>2</sup>, M. Sander<sup>2</sup>, J. Funkquist<sup>1</sup>

<sup>1</sup> Vattenfall Research and Development

Otternbuchtstrasse 14-16, 13599 Berlin, Germany

<sup>2</sup> University of Rostock

A.-Einstein-Str. 2, 18059 Rostock, Germany

<sebastian.meinke@vattenfall.de> <moritz.huebel@uni-rostock.de>

## Abstract

Offering services to stabilize the electrical grid is nowadays one of the major tasks of fossil power plants and also of significant economical relevance. However the effects on the power plants regarding the additional wear of components is uncertain. Usually the effects regarding control reserves, especially primary control occur with high frequencies and small amplitudes, which makes investigations based on measurement data impossible since the effects are masked by the noise of normal operation. In order to investigate this issue, a detailed model of a lignite power plant has been used, which was developed in Modelica for simulating and comparing scenarios with and without offering primary control reserves. The model comprises the entire water-steam cycle including turbines, preheaters and pumps, as well as a very detailed boiler model including the air supply, coal mills, a combustion chamber, heating surfaces and piping. Furthermore the power plants control system has been implemented in a very precise way. In addition the study involves an investigation on the input signals (grid frequency) and a calculation of lifetime consumption for specific components to evaluate the effects.

*Keywords: Power Plant, Dynamic Modelling, Control Reserves, Primary Control, Lifetime Consumption*

## 1 Introduction

In addition to the mere production of electrical energy, many fossil power plants are needed to provide control services, which are necessary to operate the electrical grid. In order to stabilize the frequency of the electrical grid, the consumption has to be compensated by the production at any moment. In order to guarantee this, it is required to activate or deactivate power production within seconds. The control reserves can be categorized within three corresponding grid services - primary control, secondary control and tertiary control as described in [5]. Although the necessity of granting grid services is undisputed, the consequences for fossil power plants are still uncertain. As the market for the mere production of electrical energy is declining for conventional plants, offering grid services (e.g. primary control reserves) gets more and more relevant. In order to investigate the dynamic effects on a lignite power plant, a method is presented which uses the dynamic simulation of a complex lignite fired power plant in Modelica. Similar models of a hard coal fired power plant and a combined cycle gas turbine power plant have been developed previously for different applications (e.g. [1], [2] and [3]). The dynamic model enables the user to calculate pressures and temperatures in various locations of the power plant and therefore computing mechanical and thermal stress in specific components. The results will be used to derive lifetime consumption and evaluate the effects of offering grid services, e.g. primary control reserves for this power plant.

## 2 Primary Control Requirements

The setpoint of the frequency in the European grid system is 50 Hz. The allowed variation of the mains frequency in normal mode is between 49.8 Hz and 50.2 Hz. If a failure occurs on the consumer or producer side the frequency can change within a few seconds. The primary control counteracts this. In figure 1, a typical trend for the mains frequency of one day is shown.

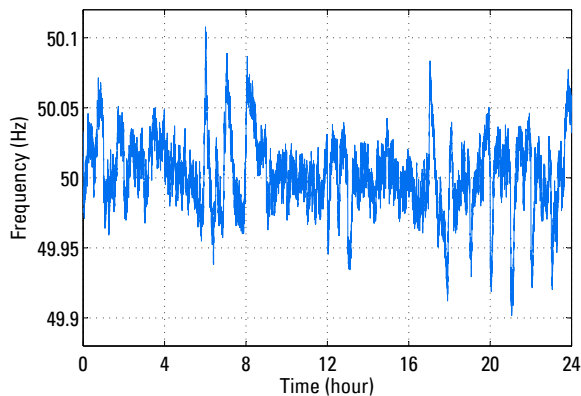


Figure 1: Trend of mains frequency of one day

Every power plant involved in the primary control has to provide at least 2 % of its effective power as primary control reserve, which has to be available within 30 s.

The dynamical behavior of a power plant is dominated by the dynamics of the steam generator and turbine. In order to provide additional electrical power in a short period of time, needed for primary control, different modes exist:

- Throttling of HP/ IP-valve: The power plant operates in all operating points with throttled high and intermediate pressure steam valves. This grants that the power plant can increase its output by opening the valves for a positive output demand and decrease its output by increasing the throttling of the valves.
- Low pressure preheater bypass: In this mode the power plant increases its electrical output by throttling the valves to the low pressure preheaters and the feedwater tank. In order to avoid thermal stress in the low pressure preheaters the speed of the condensate pump is decreased. The whole process leads to more steam in the turbine stages and thus to more electrical output. The limits of this mode are

the filling levels of the condenser and the feedwater tank. For a negative power demand the HP/IP steam valves are throttled as described in the previous point.

- High pressure preheater bypass: This mode is the same like the low pressure preheater bypass but uses the high pressure preheaters and the feedwater pump instead.

In order to investigate the influence of the primary control for a whole year, it is possible to simulate the whole year with the mains frequency as a model input, but it takes a lot of time. The implemented model has a ratio of real time to simulated time of about 1:1 to 1:10. Thus the simulation of one year would take at least one and a half month. In order to reduce simulation time the data of the mains frequency is analyzed and subdivided into characteristic signals, distinguishing between:

- changes in mains frequency, which occur every full hour, because of the changes of the power plants schedules
- noise due to the fluctuating consumer load
- power plant outage due to technical issues

These different signals will be extracted and analysed from the mains frequency data of one year. In the following, the changes in mains frequency every full hour are explained in more detail, because they have the highest amplitudes of all the characteristic signals. The basic strategy is to cut out the time data of the mains frequency and to classify these signals. Afterwards all signals in one class are averaged in order to get one signal for every class. For this purpose it must be ensured that the extremum of all the signals in one class are situated at the same time. This is realized by extracting the signals  $\pm 5$  minutes around the extremum, which can be in an interval of  $+7.5$  minutes after every full hour. Figure 2 shows all signals belonging to class 1, which contains such mains frequency changes of the year 2011 with amplitudes between  $-0.14$  Hz and  $-0.105$  Hz. Furthermore the averaged signal is shown. In order to consider the different intervals  $tn$  between the extremum and the full hour, this value is allocated to every extracted and classified signal. All signals in one class of amplitude are different in  $tn$ . An average  $tn, mean$  is calculated in order to avoid a multiplicative increase of simulation scenarios.

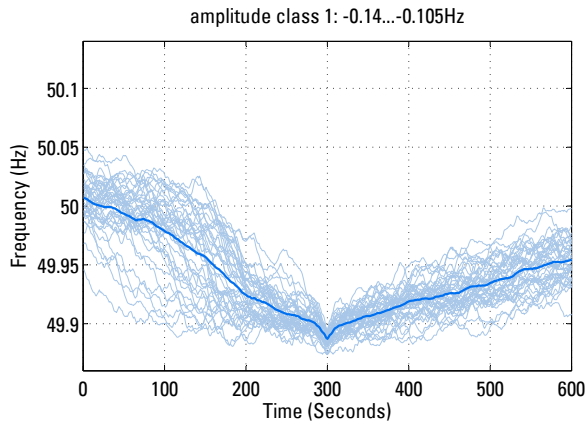


Figure 2: All signals of one class with the averaged signal in dark blue

The averaging of all mains frequency signals in one class minimizes the noise. In order to consider this noise a characteristic noise signal is added to the averaged signals of changes in mains frequency every full hour. The results of the classification are eight frequency signals for the changes of frequency every hour which are shown in figure 3. The eight frequency signals are considered as rep-

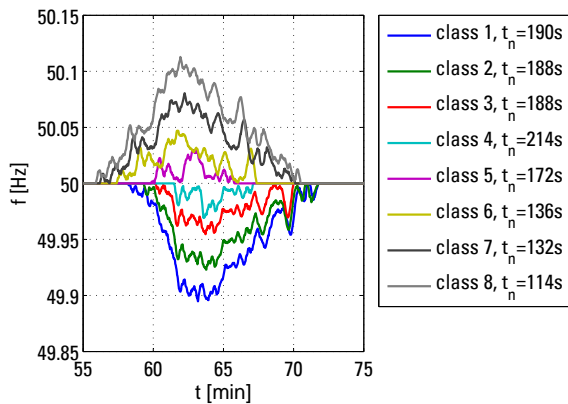


Figure 3: Scenarios of mains frequency changes which occur after every full hour. The extremum is situated  $t_n$  seconds after the full hour.

resentative inputs for subsequent investigations for all power plants participating in primary control in the ENTSO-E grid.

### 3 Power Plant System Model

#### 3.1 Reference Power Plant

The investigated power plant is a duo-block lignite power plant as shown in figure 4. Each of the blocks has two boilers, which provide steam

	Mono	Duo
Gros el. Output	265 MW	530 MW
Load Gradient	4 MW/min	8 MW/min
Primary Control	12.5 MW	25 MW

Table 1: Reference data of the Duo-Block Plant

for a mutual turbine. The block can be operated with both boilers (duo-operation) or using only one (mono-operation). One turbine has an electric output of about 530 MW. Coming from the turbine, the steam is condensed in two parallel condensers. Afterwards there is one common line of four low pressure preheaters. Then, the flow splits up for two feedwater tanks and parallel feedwater pumps and three high pressure preheaters. From there the flow enters the boilers. The boilers are using a forced circulation for the evaporation part and are equipped with parallel lines in the superheater section.



Figure 4: Investigated reference power plant

The investigated plant currently operates in base load, producing a significant amount of electrical energy. In addition to that, it offers considerable amounts of control reserves, especially primary control reserves, see Table 1 for reference data.

#### 3.2 Dynamic Process Model

The dynamic model has been built within Dymola using the open programming language "Modelica". The components used to build the model are mainly based on the "ThermalPower Library" developed by Modelon AB.

As the block arrangement in the real plant, the dynamic model consists of several components involving a diversity of different physics. An overview is given in figure 5. The general approach for all models involves the balance equations for mass and energy as well as a simplified momentum

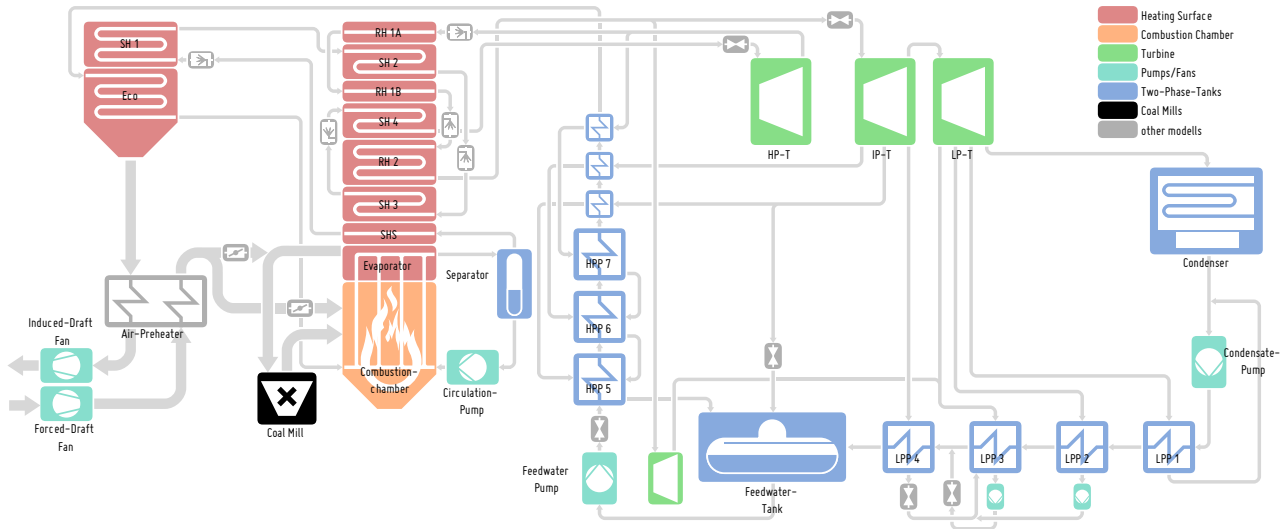


Figure 5: Implemented components of reference power plant

equation to calculate pressure drops. Using these equations as well as specific heat transfer assumptions for conduction, convection and radiation and the fluid properties for the involved mediums (flue gas, water) a power plant process can be described on a fundamental basis. A detailed explanation of physical backgrounds for all the basic models used here can be found in [6]. However the complex power plant system required some more sophisticated models which needed to be developed in order to reproduce the plants behaviour in an accurate way. One example for such a component is the lignite coal mill as presented in figure 6. The coal mill is not only responsible for grinding the coal to the desired size, but also for drying the lignite, as the water content of the fuel is usually between 50-60 %. As those effects have a significant impact on the overall process dynamics, a model had to be developed to describe the dynamic behaviour of the coal mills.

The coal mill model consists of three main paths. The gas path describes the hot flue gas which is re-circulated from the combustion chamber. Furthermore fresh air with lower temperature is added to control the temperature in the classifier of the mill. The ventilation effect of the mill is represented by a simple fan model based on the specific characteristic of the mill. The water path represents the water content of the coal which is evaporated in the mill. The energy used for evaporation is taken from the hot flue gas. After evaporation, the water is mixed with the flue gas. The coal containing a residual water content of 10-20% is represented by

the coal path.

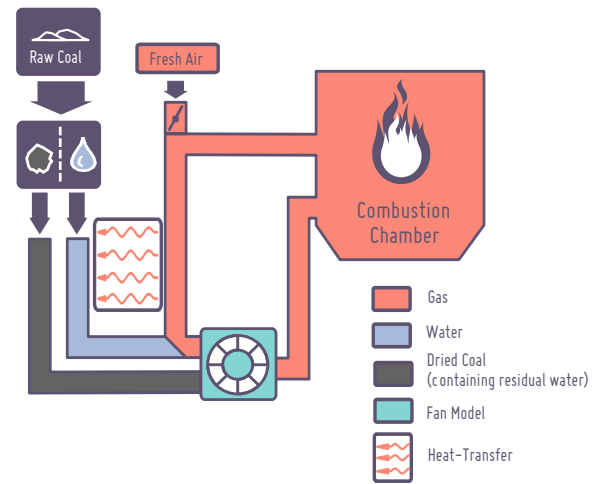


Figure 6: Schematic of the lignite coal mill model

One essential part of the model is the changed lower caloric heating value due to the evaporation of the water. In order to describe this, a simple assumption based on [7] leads to reasonable results:

$$CV_x = \frac{CV_0}{(1 - X_{A,0})(1 - X_{W,0})} (1 - X_{A,x})(1 - X_{W,x}) \quad (1)$$

Wherein  $CV$  denotes the lower caloric heating value and  $X$  the mass content of a specific component (index  $A$  for ash,  $W$  for water). The indices  $x$  and  $0$  are representing the state before and behind the evaporation stage. The delay time for



the grinding of the coal has been identified by fitting the heat release to the measurement data. [8] gives some values for the delay time in each stage of the mill, which gives a reasonable starting point for this optimisation procedure. The water steam cycle has been adapted to a single boiler. For the model, components which are actually used by both boilers are represented as a symmetric part with the half size. This concerns the models for the steam turbine, as well as the low pressure preheater line.

### 3.3 Control System

For making simulation-based statements about the influence of different power plant operation modes the thermodynamic model is coupled to a reduced copy of the original power plant control system, which is implemented using the Modelica Standard Library components. The implemented control system uses the currently calculated physical values (i.e. live steam parameter, generated power at a specific coal input) and in a consequence adjusts set values (e.g. live steam pressure) and manipulated variables (e.g. position of the feed water valve) for the water-steam cycle. Because of this feedback the accuracy and the level of details of the modelled processes needs to be reasonably high. Figure 7 is showing the hierarchical structure of the control system, which has been coupled to the process model.

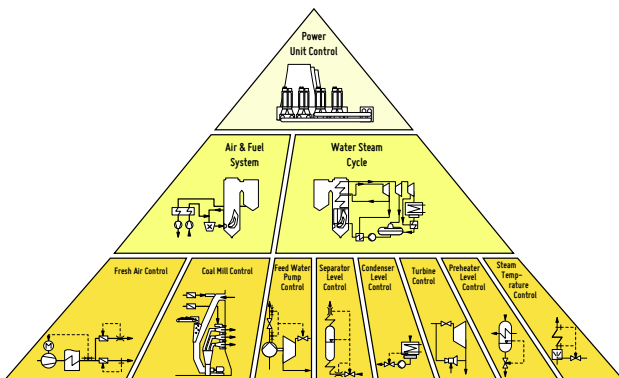


Figure 7: Overview on the control loops implemented in the model

Of particular interest is the Power Unit control which processes the incoming power and control reserve requests to a corresponding desired firing power. In detail the power plant control system sets the firing power using a map based pilot control. The expected electric power output is pre-

dicted by a transfer function based model of the firing process and the heat transfer in the boiler. The difference between this predictive value and the corresponding measurement is adjusted via a corrective control loop, as described in the VDI/VDE guideline 3508 [9]. Due to the slow transient response of the firing process, fast changes in the power output, as requested for primary control reserve, could only be conducted by using the steam storage of the boiler by altering the pressure set point until the firing process could catch up. The extracted energy needs to be filled up with a temporarily oversteered firing set point. In case of negative primary control request the pressure set point is increased until the steam production of the boiler could be reduced. By undershooting the firing power the stored steam mass could be discharged, while holding the desired electric power set point. This functionality is shown in figure 8.

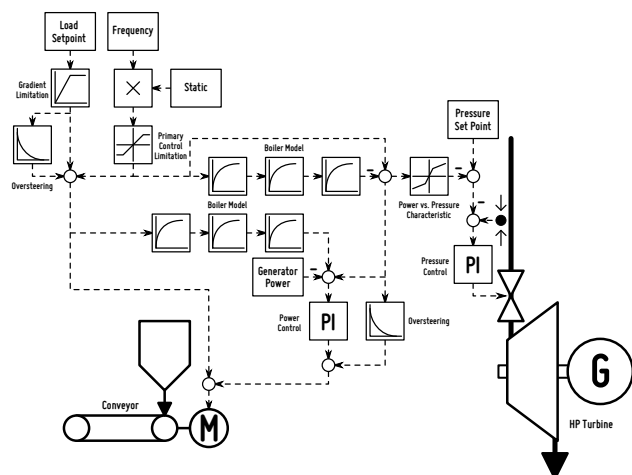


Figure 8: Schematic of the Power Unit Control

### 3.4 Validation of the Model

In order to validate the model, a scenario has been simulated covering the entire load range of the power plant from minimum load to nominal load. As required by the primary control investigation, the model has been operated in duo-operation, assuming both boilers in synchron operation.

In the model as well as in the power plant, the electrical gross output is controlled. Figure 9 shows the input schedule (green) which is used in both - the model and the real plant. The generator output (blue) is shown for the simulation (light color) and for the measurement (dark color). As in the real plant, one of the five mills in operation

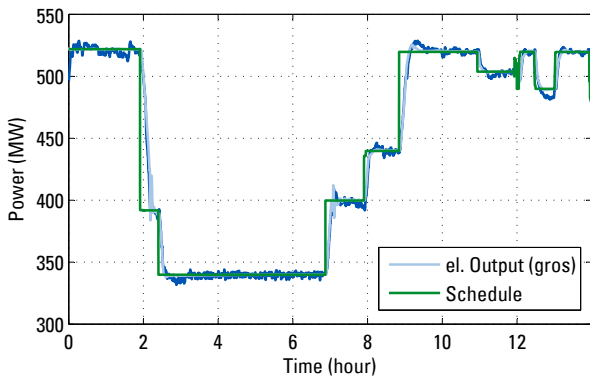


Figure 9: Schedule of the reference scenario (green), validation of power output (measurement dark blue, simulation light blue)

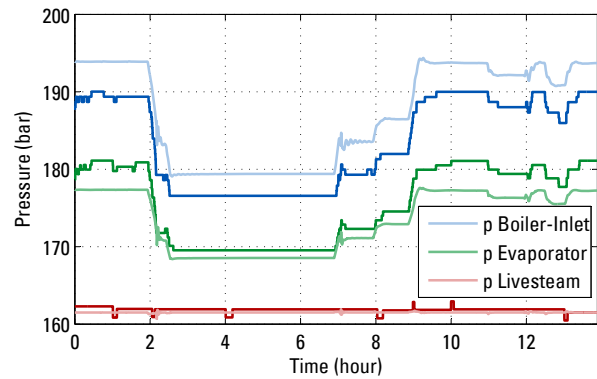


Figure 11: Validation of boiler pressure (measurement dark, simulation light)

is shut down in very low load (at about 2.5 h), this mill is returning into operation at about 7 h, causing highly dynamic effects on the entire system. As can be seen from the plot, the simulated power generation fits the measurement not only in steady state points, but also for ramps and most of the dynamic oscillations.

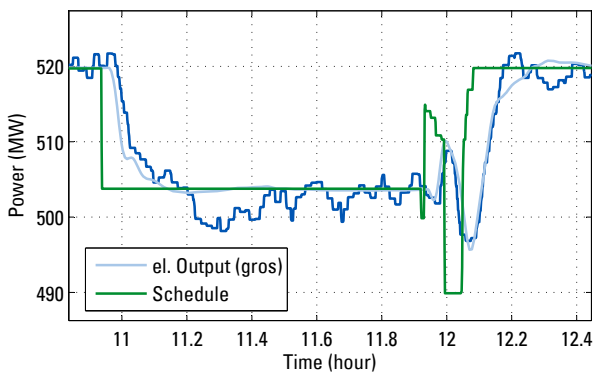


Figure 10: Zoomed validation of power output (schedule green, measurement dark blue, simulation light blue)

In figure 10 a zoomed validation plot from the reference scenario is presented. In this detailed view, setpoint-changes with amplitudes and timescale of primary control activation is shown. It can be concluded that the models really accurately represents the power plants behaviour at fast setpoint-changes. However some effects like the steady state offset between 11.2 h and 11.8 h cannot be reproduced. A reason for that might be a changing caloric heating value of the coal in the power plant which is assumed to be constant in the model.

In Figure 11 pressures from the water-steam cycle are presented. The live steam pressure (red line) is controlled by the turbine valve to a constant value of 162 bar for the entire operation range. The pressure at the evaporator and the entry of the boiler are calculated by the model based on the pressure drop of the integrated components. As can be seen from the figure, this pressure drop is highly dependent on the load, because the flow of the steam changes accordingly. Comparing measured and simulated pressures, the model shows a high agreement regarding the pressure levels.

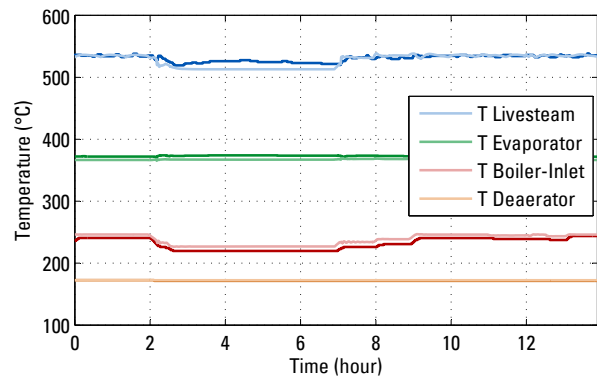


Figure 12: Validation of the temperatures (measurement dark, simulation light)

In Figure 12 some of the process temperatures are compared to the measurement data. The temperature in the deaerator is controlled to 174°C for the entire operation range. The temperature of the boiler-inlet is mainly dependent on the heat transfer in the high pressure preheaters. The temperature level in the evaporator is dependent on the pressure level, as there are two-phase conditions. The live steam temperature is being controlled by

the spray attenuators, however in very low load (between 2.5 and 7 h) the steam temperatures drop due to a shifting distribution of heat flux in the boiler and are out of their controlled range.

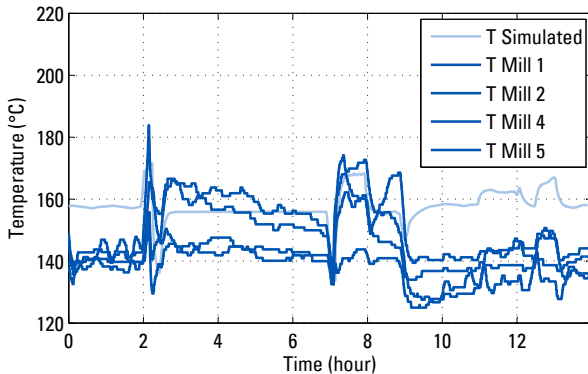


Figure 13: Validation of the coal mill model (measurement dark, simulation light)

A validation of the coal mill model based on the classifier temperatures is being presented in Figure 13. This temperature is very sensible as it represents the mixture of very hot recirculated flue gas from the combustion chamber, preheated fresh air and coal entering from ambient conditions. The simulated temperature is based on an aggregated mill model, which represents all the mills in operation, being able to switch on or off separate mills by changing the fan characteristics. The model is therefore being compared to all the mills which are permanently in operation for this scenario. It can be observed, that the average dynamic behaviour of the mill is represented quite well.

After an extensive validation based on measurement data, the dynamic model is used to calculate temperatures and pressures in critical components, thus enabling a calculation of thermal and mechanical stress according to structural mechanical approaches which are necessary derive lifetime consumption for specific simulation scenarios.

## 4 Structural Mechanics

The primary and secondary control leads to an increased number of load cycles of pressure and temperature changes within the power plant. As a result the fatigue of a component becomes more important than the creep fatigue. Particularly thick-walled pressurized components are affected by this

cyclic loading. For this reason, a routine in Matlab was developed in accordance with DIN EN 12952 [4], to determine the component stress and the lifetime until the detectable cracklength is reached. Figure 14 shows the schematic workflow of the lifetime determination until the detectable crack initiates.

Based on the pressure and temperature gradients the occurring stresses are determined in a first step. The mechanical stress  $\sigma_{\text{tang p}}$  is calculated with the boiler formula as follows.

$$\sigma_{\text{tang p}} = \left\{ \begin{array}{l} \alpha_m \cdot \frac{d_{\text{ms}}}{2 \cdot e_{\text{ms}}} \cdot p \quad \text{cylindrical shells} \\ \alpha_{\text{sp}} \cdot \frac{d_{\text{ms}}}{4 \cdot e_{\text{ms}}} \cdot p \quad \text{spherical shells} \end{array} \right\} \quad (2)$$

In addition to the pressure difference between internal and external pressures, the notch factors  $\alpha_m$  and  $\alpha_{\text{sp}}$  are considered. The notch factor depends on the geometry of the cylindrical shell and of the pipe nozzle. The thermal stress  $\sigma_{\text{ws}}$  is calculated from the temperature difference  $\Delta T$  between the two extreme values of the temperature-time function, the material factor  $\Phi_{\text{ws}}$  and the notch factor  $\alpha_T$  according to the following equation:

$$\sigma_{\text{ws}} = -\Phi_{\text{ws}} \cdot \Delta T \cdot \alpha_T \quad (3)$$

The material factor contains the elastic modulus, the Poisson's ratio and the coefficient of thermal expansion. The total stress is generated by superposition of the mechanical and thermal stress and subsequently counted with the rainflow counting [10].

In a next step, the damage  $\Delta S_i$  is calculated for each cycle (red frame in figure 14). For this, the cycles is converted to an  $R$ -ratio of  $R = -1$  using the Gerber parabola and then multiplied with a roughness coefficient and a temperature coefficient. By adding up each part the amount of damage could be obtained.

## 5 Results

In the following the results for the "class 1" scenario described in section 2 are shown. In order to investigate the influence of primary control on the life time reduction, a minus 25 % load change with and without primary control are compared with each other.

The strongly stressed components are the thick-walled outlet headers. Therefore, the headers at

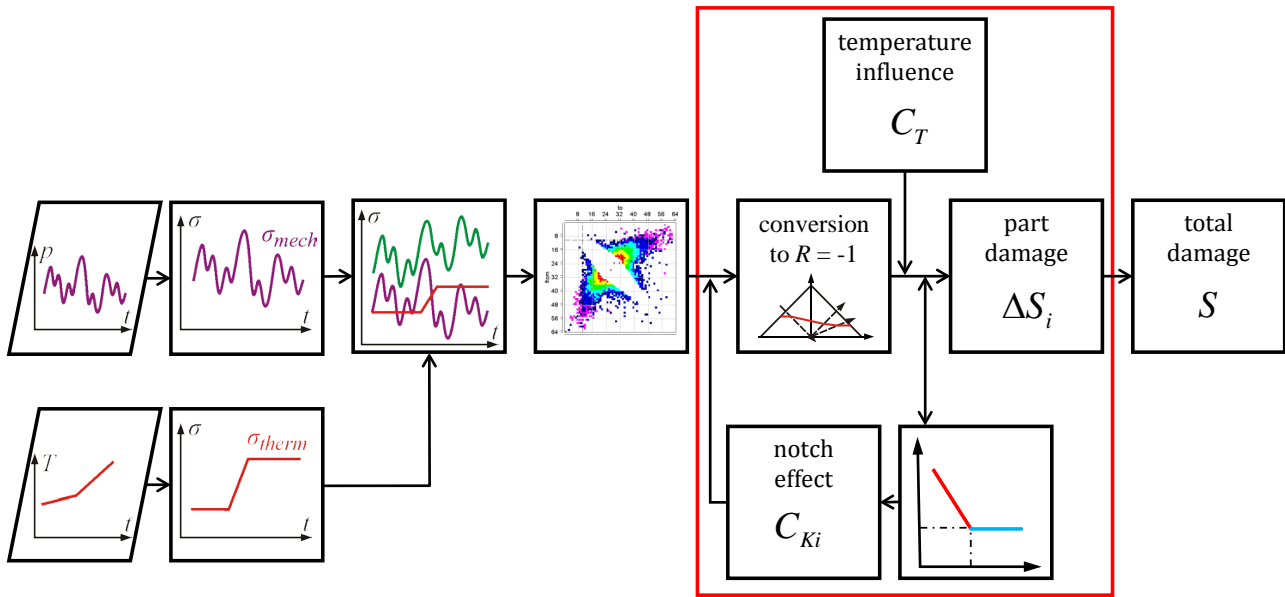


Figure 14: Schematic workflow of the lifetime determination

the outlets of superheater 4 and reheater 2 are considered first. Figure 15 shows the simulated pressure and temperature changes for the header of reheater 2. The solid lines represent the reference,

control can be seen. In temperature the impact persists longer. The entire temperature response is affected and in fact the total minimum is higher for the case without primary control.

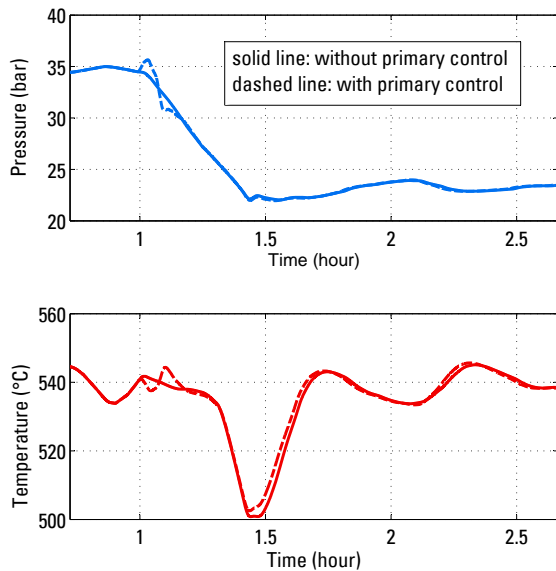


Figure 15: Simulation results for pressure and temperature of the steam in the header of reheater 2.

which is the load change without primary control, the dashed ones represent the scenario with the primary control demand. In the first 10 minutes after the load change at  $t = 1$  h, in both pressure and temperature, the influence of the primary con-

For the specific geometry of the reheater header the mechanical, thermal and total stresses are calculated as described in section 4 and the results are shown in figure 16. While the mechanical

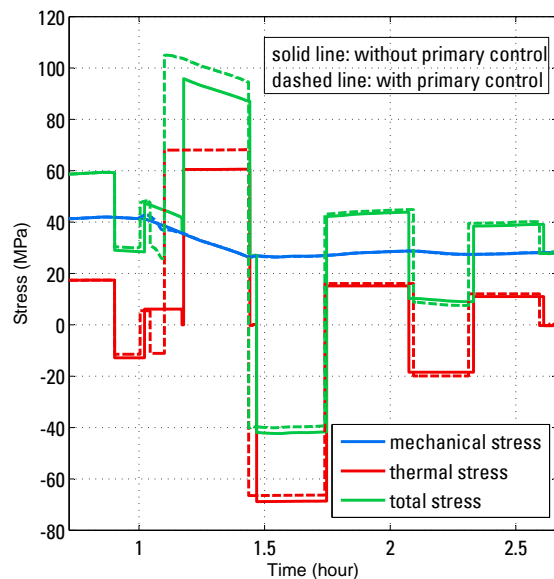


Figure 16: Mechanical, thermal and total stresses for the outlet header of reheater 2 (geometry of a cylindrical shell with a pipe nozzle).

stresses are nearly the same, the thermal stresses show a small difference due to primary control and therefore also the total stresses. In order to calculate lifetime consumption, the continuous dynamic processes have to be partitioned into discrete load cycles. Therefore the method of rainflow-counting is used as described in the previous section. The outcome is shown in figure 17 for the header of reheater 2. The red line represents the limit for the

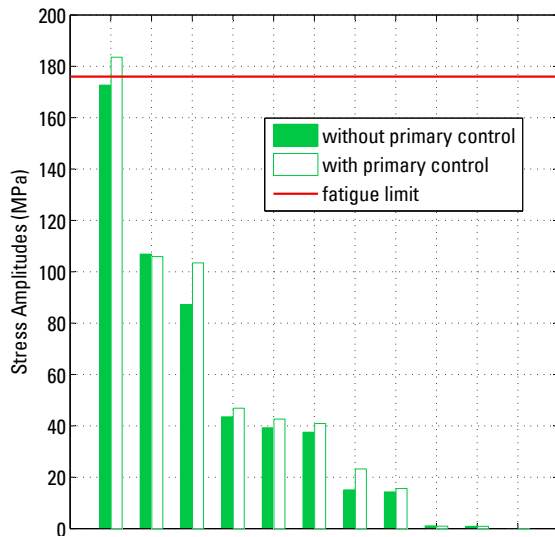


Figure 17: Sorted stress amplitudes of the outlet header of reheater 2 and the fatigue limit for this component for the scenario without and with a primary control demand.

fatigue. Obviously this fatigue limit is exceeded for the header of reheater 2 in the case of the primary control scenario. The corresponding damage  $\Delta S$  is  $1.3 \cdot 10^{-8}$ , which is determined with the S-N-curve (Wöhler curve) of the header material. In the case of the reference scenario without primary control the amplitudes are smaller and the fatigue limit is not exceeded.

For the same scenarios the stress amplitudes of the outlet header of superheater 4 are presented in figure 18. Here a damage occurs for both cases without and with primary control and is about  $1.4 \cdot 10^{-6}$  and  $6.8 \cdot 10^{-7}$ , respectively. In contrast to reheater 2 the primary control scenario leads to smaller stress amplitudes than the pure load change.

In conclusion the results show that the impact of primary control on the life time reduction is not only determined by the pressure and temperature conditions but also strongly depends on the

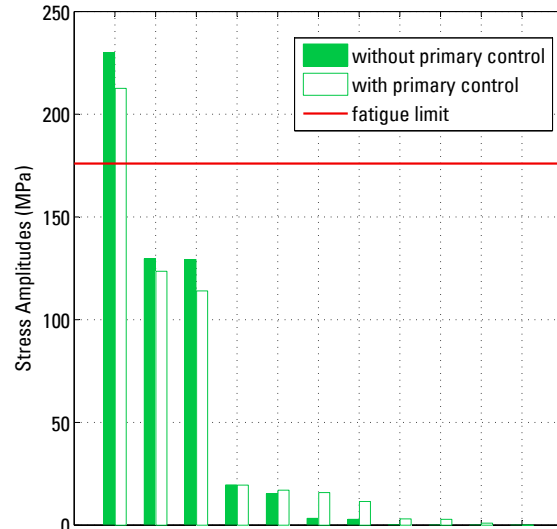


Figure 18: Sorted stress amplitudes of the outlet header of superheater 4 and the fatigue limit for this component.

geometry of the component. Thus the structural mechanics calculations need to be done for all components of the power plant. Furthermore, several scenarios have to be investigated and connected with their annual occurrence to identify the overall damage of the power plant due to primary control.

## 6 Summary

A very comprehensive dynamic model of a lignite fired power plant has been developed as presented in this paper. The model allows a diversity of applications focussing on dynamic operation e.g. control optimisation, calculation of lifetime consumption or energetic optimisation. One example application is the ongoing investigation on the impact of primary control on this power plant. The result of a first scenario has been presented and explained on the example of two highly stressed components. In the next stages of this investigation, all the scenarios presented in section 2 will be simulated and multiplied with the share of their appearance in order to assess the overall impact of primary control.

## 7 Acknowledgement

The authors would like to thank VGB PowerTech and Vattenfall for providing financial support and input data as well as the participating delegates for

inspiring discussions and helpful hints from practical experience during the project meetings. Also we would like to thank all the people at the power plant for supporting the acquisition of design and process data.

## References

- [1] Meinke, S., Gottelt F., Müller, M., Hasse E., Modeling of Coal-Fired Power Units with ThermoPower focussing on Start-Up Processes, 8th Modelica Conference, Dresden 2011
- [2] Brunnemann, J., Gottelt, F., Wellner, K., Renz, A., Thüring, A., Roeder, V., Hasenbein, C., Schulze, C., Schmitz, G., Eiden, J., Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO<sub>2</sub> Capture, 9th Modelica Conference, Munich, 2012
- [3] Meinke, S., Modellierung thermischer Kraftwerke vor dem Hintergrund steigender Dynamikanforderungen aufgrund zunehmender Windenergie- und Photovoltaikeinspeisung, Dissertation, Rostock, 2012
- [4] DIN EN 12952-3 Wasserrohrkessel und Anlagenkomponenten - Teil 3: Konstruktion und Berechnung für drucktragende Kesselteile. Deutsche Fassung EN 12952-3, 2011.
- [5] TransmissionCode 2007 Netz- und Systemregeln der deutschen Übertragungsnetzbetreiber, Verband der Netzbetreiber, 2007
- [6] Thermal Power Library Documentation, Modelon, 2013
- [7] O'Kelly, P., Computer Simulation of Thermal Plant Operations, Springer, 2013
- [8] Effenberger, H., Dampferzeugung, Springer 2000
- [9] VDI/VDE-Guideline 3508, Unit control of thermal power stations, VDI/VDE, 2003
- [10] Matsuishi, M., Endo, T., Fatigue of metals subjected to varying stress, Japan Soc. Mech. Engineering, 1968

# Use of External Fluid Property Code in Modelica for Modelling of a Pre-combustion CO<sub>2</sub> Capture Process Involving Multi-Component, Two-Phase Fluids

Carsten Trapp<sup>\*1</sup>, Francesco Casella<sup>2</sup>, Teus van der Stelt<sup>1</sup> and Piero Colonna<sup>1</sup>

<sup>1</sup>*Delft University of Technology, Propulsion and Power, Kluyverweg 1, 2629 HS Delft, The Netherlands*

<sup>2</sup>*Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Via Ponzio 34/5, 20133 Milano, Italy*

## Abstract

This paper presents the development of a system model for a pre-combustion CO<sub>2</sub> capture process as part of an integrated gasification combined cycle power plant. This process entails the modelling of highly non-ideal, two-phase multi-component mixtures which are currently not supported by available Modelica media libraries or interfaces.

Therefore, an interface prototype was developed and tested for the modelling and simulation of the CO<sub>2</sub> capture process. Limitations concerning the modelling approach and improvements targeting the computational efficiency are discussed. Recommendations about the design of a library for the use of external property estimation code in Modelica conclude the treatment.

*Keywords:* pre-combustion CO<sub>2</sub> capture; two-phase, multi-component fluids; external fluid property code

## 1 Introduction

Pre-combustion CO<sub>2</sub> capture applied to integrated gasification combined cycle (IGCC) power plants is a promising technical solution to mitigate CO<sub>2</sub> emissions and therefore the effect of climate change [1]. The integration of the CO<sub>2</sub> removal unit with the very complex gasification process and combined cycle power plant leads to challenges especially regarding dynamic operation. Nowadays, dynamic performance of fossil-fuelled power plants becomes increasingly important as the share of electricity produced by renewable energy sources, which is inherently unsteady, is continuously growing. Therefore, the inte-

grated capture process has to be able to follow frequent and fast load changes to allow for flexible power production. In order to study the transient performance of the pre-combustion CO<sub>2</sub> capture unit during load variations, dynamic models of the entire system and models of the individual components have thus been developed using the Modelica modelling language. The models have been validated by comparison with experimental data obtained from a unique, fully instrumented CO<sub>2</sub> capture pilot plant, which has been realized at the Buggenum IGCC power station in the Netherlands by the utility company Vattenfall [2].

The main challenge of the model development is related to the computation of fluid properties, in particular phase equilibria, due to the fact that highly non-ideal, two-phase, multi-component fluids are involved in the capture process. Currently, Modelica medium models are not available for this type of fluids.

One possibility is to implement required medium models, just as the process models, in the Modelica language, with the advantage to be able to perform efficient simulations as the code can be optimized, provided the equation of state is written in a declarative way, which might not always be possible. However, the implementation of non-ideal fluid property models is rather time-consuming and not trivial, as dedicated solution algorithms might be required for efficiency and numerical robustness.

The other possibility is to make use of available thermophysical property packages and interface these tools with Modelica. Employing external tools for the computation of fluid properties provides some advantages: 1) typically the property software employs dedicated algorithms for fast and robust calculations of the fluid properties, 2) the property package can be interfaced with a wide variety of engineering software tools (e.g. steady-state and dynamic system mod-

<sup>\*</sup>Email: C.Trapp@tudelft.nl, P.Colonna@tudelft.nl

elling, component design, CFD, etc.) allowing for the use of the same thermophysical properties, thus eliminating one common source of uncertainty, 3) a wide range of pure fluids and fluid mixtures described with suitable and accurate equation of states are available in the property package.

This concept was successfully demonstrated with the ExternalMedia library [3] which supports two-phase, single-substance fluids all compatible to the Modelica.Media interface. However, multi-component fluids as required for the CO<sub>2</sub> capture process are not supported because the interface for multi-phase fluid mixtures is limited to two-phase pure (or pseudo-pure) components.

For this reason, a prototype for an interface supporting two-phase, multi-component fluids was developed and tested for the modelling and simulation of the CO<sub>2</sub> capture process.

The objective of the work documented here is to demonstrate the feasibility of modelling such a chemical process with Modelica by making use of external fluid property code and to indicate limitations concerning the modelling approach as well as to discuss possibilities for the improvement of the computational efficiency. Finally, recommendations shall be drawn for the design of a generic interface to external fluid property code.

## 2 Pre-combustion CO<sub>2</sub> capture process

The simplified process flow diagram of the CO<sub>2</sub> capture pilot plant built at the site of the Buggenum IGCC power station is depicted in Figure 1. The syngas from the gasifier entering the CO<sub>2</sub> removal unit is mixed with process water in order to obtain a pre-set H<sub>2</sub>O:CO ratio, and then it is fully evaporated and superheated by means of electrical heaters. The carbon monoxide present in the syngas is converted into hydrogen and carbon dioxide via a three-stage, sweet, high-temperature water-gas shift (WGS) process. The excess process water is recovered from the shifted syngas through condensation and then recycled. Subsequently, the carbon dioxide is removed from the syngas in the CO<sub>2</sub> absorber by means of physical absorption utilizing the solvent dimethylether of polyethylene glycol (DEPEG). The resulting H<sub>2</sub>-rich syngas is fed to the gas turbine of the combined cycle power plant and the CO<sub>2</sub> is recovered by three-stage depressurization of the loaded solvent. The lean solvent is recycled to the absorber, while the CO<sub>2</sub>-rich product stream is compressed to a state suitable for storage.

Throughout the evaporation and condensation in

the shifting section, vapour-liquid equilibrium of the syngas-water mixture is assumed, which requires rigorous fluid property computations. The water-gas shift reaction occurs at conditions where the syngas-steam mixture can be described as an ideal gas. Throughout the absorption section, vapour-liquid equilibrium of the syngas-solvent mixture is also a verified hypothesis.

The main differences between the pilot plant and a large-scale CO<sub>2</sub> capture process concern the heat integration in the shifting section. In a large-scale design the electrical devices used for heating, cooling and condensation are replaced by heat exchangers.

## 3 Model development

The thermophysical properties of the two-phase multi-component syngas-water/syngas-solvent mixtures are calculated with the Perturbed Chain - Statistical Associating Fluid Theory (PC-SAFT) equation of state [4] due to its success in predicting vapour/gas-liquid equilibria of complex fluids and mixtures for a broad range of conditions. This EoS has been validated against experimental measurements and data from literature [5] and implemented, together with fast and robust algorithms, into an in-house property package [6] which is interfaced with the dynamic modelling tool via the ModelicaFluidProp Modelica library.

### Library architecture

The ModelicaFluidProp library provides the functional interface that allows to integrate external fluid property codes into Modelica models. The library contains two parts, the Modelica front-end which makes various functions available for the calculation of different property sets (for instance "AllProps" or "TwoPhaseDeriv") and the C/C++ back-end, containing C++ objects that carry out the interfacing between the Modelica level and the external software tool.

The Modelica library contains a generic package FluidPropMedium. The actual external fluid property code is specified by setting values to constants such as ModelName, which defines the name of the external library, nComp, which specifies the number of fluid constituents and Comp, which defines the name of the individual constituents. The external medium model can be used in any component model and is not extending any medium package from the standard Modelica library.

The implemented set of functions in the FluidPropMedium package mirrors one-to-one property functions available in the external property tool which



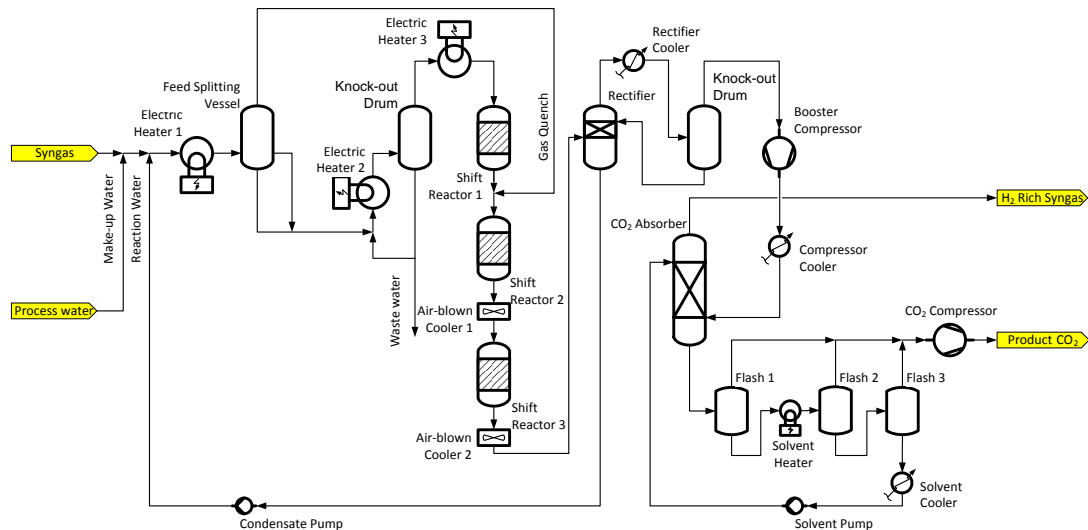


Figure 1: Process flow diagram pilot plant.

are interfaced with corresponding C-functions defined in the C interface layer.

In the following, the working principle of the library is explained, based on the exemplary code below:

```
import SI = Modelica.SIunits;

package SyngasDEPEG
  extends ModelicaFluidProp.FluidPropMedium(
    modelName="PC-SAFT",
    nComp=6,
    Comp={"carbon monoxide","hydrogen",
          "carbon dioxide","water","nitrogen",
          "DEPEG"})
end SyngasDEPEG;

model Example
  SyngasDEPEG.AllPropsOut prop;
  SI.Temperature T;
  SI.Pressure P;
  SI.SpecificEnthalpy h;
  SI.Density d;
  SI.MoleFraction Y[SyngasDEPEG.nComp]=
  SyngasDEPEG.reference_X;
  SI.MoleFraction Yliq[nComp]
  "Molar fractions of liquid phase";
  SI.MoleFraction Yvap[nComp]
  "Molar fractions of vapour phase";
equation
  (prop,Yliq,Yvap) =
  Medium.AllProps("PT",P,T,Y);
  h = prop.h;
  d = prop.d;
end Example;
```

The AllProps function of the FluidPropMedium package calls the corresponding C function of the interface and passes the specification of the thermodynamic state ("PT"), the values of pressure, temperature and composition as well as the constants for medium identification. The interface function handles the creation of an object for the external property code and the execution of the solver to compute

the required properties. The calculated fluid properties are passed via the prop record to the Modelica code. The AllProps function returns all primary thermodynamic properties such as  $P$ ,  $T$ ,  $v$ ,  $d$ ,  $h$ ,  $s$ ,  $u$ , etc. which can be computed with hardly any additional computational cost when solving the equation of state. Secondary thermodynamic properties such as heat capacity, speed of sound, various single-phase partial derivatives and transport properties are computed with a separate function as these properties are less often needed and require additional computations. The computationally expensive two-phase partial derivatives are combined in another property function.

The arrangement of primary and secondary fluid properties in meaningful functions allows for a flexible use and avoids unnecessary repeated computations.

## Process models

The objective is to develop physical-based component models which allow the modelling and simulation of the pre-combustion CO<sub>2</sub> capture process. The model structure shall facilitate the integration of external functions for thermophysical property calculations.

For the modelling of the CO<sub>2</sub> capture process various component models are required. Whenever possible models were reused from available Modelica libraries. For example, basic component models such as sinks, sources, valves, pressure drops, pumps, heat exchange and flow models, are taken from the ThermoPower library [7, 8] and adapted in terms of their media models which have been replaced with functional calls to the external property tool.

For the following components new models were implemented:

- **Flash vessel**

The process of phase separation is modelled under the assumption of thermodynamic equilibrium between the liquid and vapour phase at all times. This is a justified assumption since the process is typically designed to ensure sufficient mixing. The model describes the holdup of vapour and liquid with conservation equations which account for both phases together. Saturated conditions are assumed for the liquid and vapour outlet streams and therefore entrainment of liquid in the vapour flow is neglected. The flash vessel model is implemented as a "pure storage" component and hence frictional losses are not considered. The static pressure head due to the liquid level in the vessel is accounted for in the algebraic momentum balance. Heat transfer from the fluid (both vapour and liquid phase) to the vessel wall and accumulation of thermal energy in the wall as well as heat losses to the environment are neglected. Thus, also superficial condensation is assumed to be negligible.

- **Water-gas shift reactor**

The reaction of carbon monoxide with steam to produce carbon dioxide and hydrogen is described in a lumped-parameter model. The syngas entering and leaving the reactor is an ideal-gas mixture containing CO, CO<sub>2</sub>, H<sub>2</sub>, H<sub>2</sub>O, N<sub>2</sub>. Other trace constituents are neglected. The model accounts only for the WGS reaction. Intermediate reactions involving other chemical species are neglected. The reactor model is subdivided into five sub-models (Figure 2): reaction node, mixing gas volume, convective heat transfer, thermal storage and pressure drop.

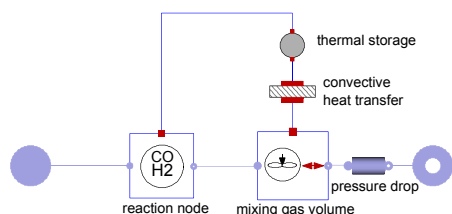


Figure 2: Object diagram of reactor component.

The WGS reaction takes place in an infinitesimally small volume (reaction node) representing one finite discretization of the catalyst and reaches thermodynamic equilibrium. The accumulation of mass and energy in the bulk phase of

the reactor are described in a perfectly mixed volume (mixing gas volume) which receives the reaction products. This control volume exchanges heat with the catalyst by means of convection. The storage model describes the accumulation of thermal energy in the catalyst. Heat transfer to the environment is neglected.

The water-gas shift reactor is discretized in axial direction by an array of reactor models in order to correctly describe the gradual changes in reactor outlet conditions during transient operation. Changes in the reactor inlet conditions reach the reactor outlet with a delay due to thermal storage in the catalyst, which cannot be represented with a 0-dimensional model due to the high number of transfer units between the gas and the catalyst itself.

However, this one-dimensional discretization does not represent the the actual axial reactor profile as equilibrium conditions are assumed in each reactor model element for simplicity. In steady-state conditions the equilibrium temperature is reached at each discretization of the catalyst, which also determines the temperature dependent WGS reaction.

- **Pilot plant specific heater and cooler components**

Various electrical components for evaporation, superheating, cooling and condensation were in particular developed for the pilot plant process and will not be part of a large-scale plant in this specific configuration. The models were developed following a modular approach such that sub-models can be reused. The models were typically subdivided, if applicable, in flow models, heat transfer models and thermal storage models. Whenever possible models from the ThermoPower library were used, typically in case the medium was water or ideal gas, or adapted.

- **Absorption column**

The packed column model for physical absorption (no chemical reactions) is discretized in theoretical stages in axial direction and counter-current flow of the vapour and liquid is assumed. Each stage is modelled by an equivalent tray module ("pure storage") and a resistive module. By connecting a series of storage and resistive modules (Figure 3) a low index of the equation system can be maintained (detailed discussion see Section 4).

In the equivalent tray module, pressure, temperature and composition of the liquid and vapour

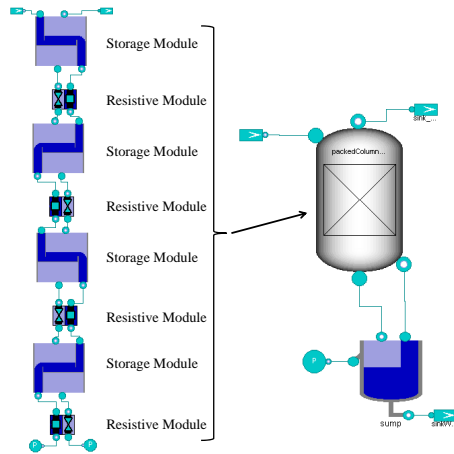


Figure 3: Model structure of the absorption column.

phase are determined by solving the conservation equations for mass and energy assuming thermodynamic equilibrium between liquid and vapour [9]. This module is based on the flash vessel model with its assumptions stated above. In the resistive module the momentum equation is substituted by empirical correlations to describe the hydrodynamics of the stage predicting the liquid and vapour flow rate as a function of the pressure difference between the stages, the liquid holdup and the packing characteristics. Empirical correlations for the pressure drop and liquid holdup are published by Stichlmair et al. [10] and Billet and Schultes [11].

## 4 Modelling approach / Lessons learned

The use of external fluid property functions in Modelica process models puts some restrictions on the model development. Specific attention requires the formulation of the differential model equations, the choice of state variables and the causality of the system model. These issues are addressed in the following.

### 4.1 Choice of state variables

For dynamic modelling of thermo-physical systems not only the choice of the system state variables is of importance but also the selection of the thermodynamic states used to determine fluid properties. In general, the system state variables should allow for an easy computation of the thermodynamic properties required to determine the system performance and the thermodynamic state variables should unambiguously determine the fluid state. Further, the choice of sys-

tem state variables can have a significant influence on ease of initialization, numerical robustness and computational speed.

In the following, three different possibilities for the choice of state variables and their influence on resulting differential and algebraic equation (DAE) system as well as simulations speed are analysed. As an example, dynamic mass and energy balances for describing storage of vapour and liquid in a volume under the assumption of thermodynamic equilibrium are used. These equations can be found in the flash vessel and absorber tray models.

### Explicit system state variables $M, u, X_i$

In the most simple way the dynamic mass and energy balance can be formulated as

$$\frac{dM}{dt} X_i + M \frac{dX_i}{dt} = w_{in} X_{in,i} - w_{liq} X_{liq,i} - w_{vap} X_{vap,i}, \quad (1)$$

$$M \frac{du}{dt} + u \frac{dM}{dt} = w_{in} h_{in} - w_{liq} h_{liq} - w_{vap} h_{vap}, \quad (2)$$

where  $M$  is the total mass,  $u$  is the internal energy,  $X_i$  is the component mass fraction vector whereby the subscript  $i$  ranges from 1 to the number of species in the mixture,  $w$  is the mass flow rate and  $h$  is the specific enthalpy. A formulation purely based on moles is also possible and might be the preferred choice as conversions between mole and mass are avoided.

Assuming  $M, u$  and  $X_i$  are selected as state variables and no other variables than the states appear under the time derivative the system solution can be obtained straightforward. This applies, for example, to a single flash vessel model where the index of the DAE system is 1. Section 4.2 covers the case of higher index problems, e.g. when dealing with a model of two flash vessels connected with a zero pressure drop.

The mass and energy equations represent a compact and very declarative formulation of the conservation equations. The pressure  $P$ , the temperature  $T$  and the mass composition  $X_i$  are chosen as thermodynamic states, therefore

$$prop = prop(P, T, X_i). \quad (3)$$

For this state selection property computations for two-phase mixtures are most efficient and robust with the used external media library (see Section 4.3). As the system states and thermodynamic states differ, implicit equations must be solved at each time step in order to determine the thermodynamic state variables.

Consequently, with the solution strategy used by Dymola the computational speed is affected by the iterations on the property function, which are anyway the most expensive part of the entire simulation.

### Implicit system state variables $P, T, X_i$

The Modelica language features the option to change states by means of the StateSelect attribute, without changing the declarative formulation of the dynamic mass and energy balance. Hence, pressure, temperature and composition can be selected as preferred states in accordance to the thermodynamic states with the aim to reduce iterations during the solution procedure. However, as the total mass and internal energy remain under the time derivative and not being states, the tool attempts to symbolically differentiate  $M$  and  $u$  with respect to the states  $P, T, X_i$  in order to establish a relationship between the old states, which have been discarded, and the newly selected states. This procedure fails as the external property call cannot be differentiated.

### Explicit system state variables $P, T, X_i$

In order to facilitate a feasible solution an explicit expression for  $\frac{dM}{dt}$  and  $\frac{du}{dt}$  needs to be provided as a function of the state variable derivatives  $\frac{dP}{dt}$ ,  $\frac{dT}{dt}$  and  $\frac{dX_i}{dt}$ . The needed time derivatives therefore read

$$\frac{dM}{dt} = -Md \left[ \left( \frac{\partial v}{\partial p} \right)_{T,X} \frac{dp}{dt} + \left( \frac{\partial v}{\partial T} \right)_{p,X} \frac{dT}{dt} + \left( \frac{\partial v}{\partial X_i} \right)_{p,T,X_{j \neq i}} \frac{dX_i}{dt} \right], \quad (4)$$

$$\frac{du}{dt} = \left( \frac{\partial u}{\partial p} \right)_{T,X} \frac{dp}{dt} + \left( \frac{\partial u}{\partial T} \right)_{p,X} \frac{dT}{dt} + \left( \frac{\partial u}{\partial X_i} \right)_{p,T,X_{j \neq i}} \frac{dX_i}{dt}. \quad (5)$$

The partial derivatives of  $u$  and  $v$  with respect to  $P, T, X_i$  need to be provided by the external media library. Two-phase partial derivatives of mixtures, which are based on properties obtained from the phase equilibrium calculations, are commonly not available among the typical thermophysical fluid properties and therefore have been specifically implemented in the framework of this model development. These mixture derivatives are currently computed numerically. Future work might consider the analytical formulation as implemented, for example, in Multiflash [12]. However, also the calculations based on analytical expressions is expected to be computational expensive [13].

These mixture derivatives cannot be fully expressed analytically and hence the numerical calculation affects the computational speed of the dynamic model.

In Figure 4 the simulation performance of a flash vessel with system states  $M, u, X_i$  and state variables  $P, T, X_i$  is compared. The simulation of the latter model requires 90 % less functional calls to the external property library promoted by the same choice for system and thermodynamic state variables. However, considering the computational time the latter model only leads to an improvement of 40 %. This is explained by the additional calculation of partial derivatives required when changing the states from  $M, u, X_i$  to  $P, T, X_i$ .

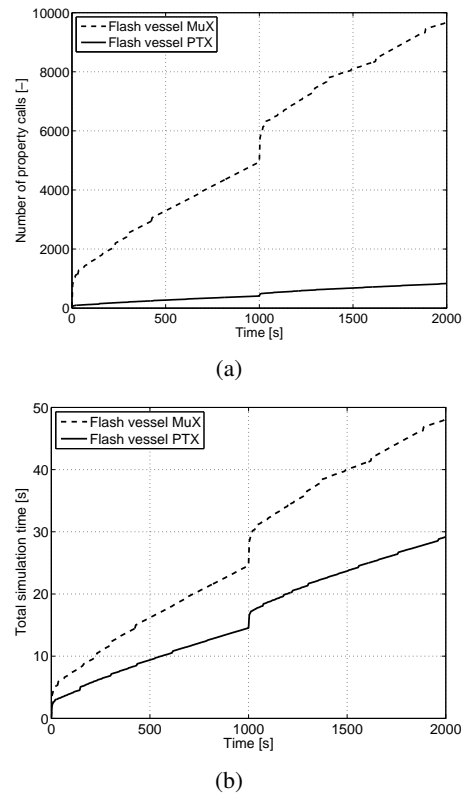


Figure 4: Comparison of flash vessel models with state variables  $M, u, X_i$  and  $P, T, X_i$ . a) Number of property calls. b) Total simulation time.

The observed difference in computational speed between both choices of state variables might also be related to the solution strategy employed by the software tool. Dymola was adopted for this project, and Dymola obtains the solution of the DAE system by iterating on a nested loop, and solving the ODE's in the outer and the algebraic equations in the inner loop. It might be possible that using a DAE solver directly on the original DAE system in this case might lead to a much smaller difference in computational speed between both solutions. However, a direct solution strat-

egy might entail more trouble during initialization and shorter time steps to achieve convergence might be required. A hybrid strategy where both approaches are applied, dynamically switched, might be interesting to explore.

## 4.2 Developing index-1 models

Considering a model containing two flash vessels, component mass and energy balance represented by equation 1 and 2, with zero or constant pressure drop, then the index of the DAE system is larger than 1, assuming  $M$ ,  $u$  and  $X_i$  are chosen as state variables.

The solution of a DAE system with higher index is commonly obtained by symbolic manipulation of the equations system in order to reduce its index to 1. Current simulation tools implementing the Modelica language employ state-of-the-art techniques for index reduction. Difficulties during index reduction might arise in case fluid property calculations must be symbolically differentiated. If the fluid correlations or equation of state are implemented as a Modelica media library, possibly accompanied with annotations to compute its time derivatives, symbolic manipulation can be performed by the tool resulting in a successful index reduction. However, in case external media libraries are used, interfaced with the Modelica models, index reduction fails as external functions cannot be manipulated. This can be resolved by either supplying any time derivatives required for index reduction or by developing the Modelica models such that the DAE system remains in the index-1 form. The latter approach has been followed for the dynamic modelling of the CO<sub>2</sub> capture pilot.

### Causal versus acausal approach

Following the acausal modelling approach, which is fully supported by Modelica, connections of sub-models might not respect the causality leading to a system of DAE's of higher index (example of two flash vessels described above). Therefore, a mixed approach is applied during the model development. The sub-models and interfaces are defined and developed in an object-oriented manner such that the models can be employed in an acausal context. However, the system has been analysed and decomposed into subsystems following the causal modelling paradigm in order to avoid the occurrence of higher order DAE systems [14]:

- Identification of bilaterally coupled variables of the models.
- Discretisation of the model in resistive and storage modules, namely solving the conservation

laws for flow and potential in different control volumes.

- Connect the resistive to the storage modules and vice versa such that potential variables are inputs of the resistive and outputs of the storage modules. Flow variables are inputs to the storage and outputs to the resistive modules.

By following a more causal development and arrangement of the models the DAE system can be maintained in the index-1 form which allows for a straightforward use of external property functions. The disadvantage of this approach is that it poses restrictions on the connection of modules. It might be necessary to include dummy modules with no process functionality in order to maintain the resistive-storage structure.

## 4.3 Improvement of computational time

Based on the current experience obtained from the modelling of the CO<sub>2</sub> capture process, the computation of thermodynamic fluid properties, in particular phase equilibria, accounts for the main share of the simulation time (in the order of 95 %). In addition, much more complex and hence computational expensive thermodynamic models are required when dealing with highly non-ideal, multi-component mixtures in comparison to fluid models for water or ideal gas. Therefore, an appropriate and smart use of property functions should be considered during the entire model development process. In the following, suggestions are presented, which might significantly contribute to a successful convergence and more efficient simulations.

### Choice of thermodynamic states

For an efficient use of external property functions in Modelica, it is necessary to have knowledge on the property calculations performed in the external tool. The thermodynamic state can be determined for different choices of independent variables, for example "PT", "Pv", "Ph", "Pq", "Tv", "Tq", "uv", etc., where  $P$  is the pressure,  $T$  is the temperature,  $v$  the specific volume,  $h$  is the specific enthalpy,  $q$  is the vapour quality and  $u$  is the specific internal energy. In case of mixtures the component fraction vector  $X_i$  is added to set of independent variables.

The external tool employs an isothermal ("PT") or an isenthalpic ("Ph") flash algorithm developed by Michelsen [15] for the PC-SAFT and cubic EoS's, which are robust and reliable equilibrium calculations based on the minimization of Gibbs free energy. If other thermodynamic states are chosen as input, then

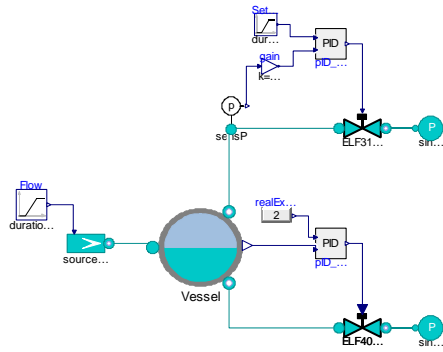


Figure 5: Object diagram of a flash vessel with vapour and liquid control valves.

the solution is obtained by iteration on either the PT- or Ph-flash calculation. Any computations including the vapour quality  $q$  as input use a bubble/dew point calculation, which is much more difficult to perform and is far less robust than the flash algorithms.

To conclude, "PT" or "Ph" are recommended as thermodynamic states as these inputs allow for fast and robust fluid property calculations.

### Single and two-phase property calculations

The flash algorithm first determines the vapour and liquid composition of the fluid (computation of phase equilibrium), which indicates if the fluid state is in the two-phase or the single phase region. Then the vapour and liquid properties such as  $v$ ,  $h$ ,  $s$ , and  $u$  are computed. If applicable two-phase properties are calculated based on the single phase properties using appropriate mixing rules. The initial determination of the liquid and vapour composition is computationally very expensive and hence this step should be omitted if it is known a-prior that the fluid is present in the single phase. Therefore, the option of skipping the flash calculation and just performing a single phase calculation was implemented by using, for example, "PT-1ph" as input specification.

The flash vessel component is one example where this ability to explicitly indicate the fluid phase finds application. For the conditions in the vessel two-phase properties are required whereas in valves connected to the vapour and liquid outlet single phase properties are sufficient. Experience has shown, that by using single phase property computations where applicable throughout the process the simulation time can be reduced significantly (see Table 1).

### Redundant property calls

The model of a simple phase separation as depicted in Figure 5 provides another example on how the com-

putational efficiency can be improved by optimizing the property calculations. The flash vessel model contains a property call which determines required primary two-phase, liquid and vapour properties, such as  $h_{2ph}$ ,  $d_{2ph}$ ,  $h_{liq}$ ,  $d_{liq}$ ,  $h_{vap}$  and  $d_{vap}$ ,

$$prop_{2ph,liq,vap} = prop(P, T, X_{i,2ph}). \quad (6)$$

In the valve model connected to the vapour outlet of the flash vessel, the vapour density  $d_{vap}$  is required to close the set of conservation equations by

$$prop_{vap} = prop(P, T, X_{i,vap}). \quad (7)$$

It is obvious that, under the assumption of adiabatic operation and no frictional losses, both property computations provide the same result for the vapour density. However, the property calls are different due to the fact that in the vessel the two-phase mixture composition and in the valve the vapour composition is used as input. During the process of translating the model into a set of solvable equations, the compiler will not realize that the second property call is in principle redundant.

One solution to overcome this issue is to transfer the required density via additional output and input connectors from the flash vessel to the valve. This solution has been implemented as optional choice, which can be activated by the modeller. Another possibility would be the use of conditional connectors which transfer next to the flow and potential variables possibly an array of all fluid properties.

The improvement in computational time is summarized in Table 1. In comparison to the use of single phase property calls in the vapour valve the improvement is rather small. However, the aim is not to obtain the most efficient simulations but to demonstrate a modelling approach which might contribute to efficiency.

### Approximation of two-phase partial derivatives

One of the most time-consuming part of the property calculations is the computation of two-phase partial derivatives, which are for example required in the flash vessel component and absorber tray model (see equations 4 and 5)

The results of the partial derivatives are used in the mass and energy balances. Various tests indicated that if the change in the absolute value of the thermodynamic states during the simulations is small the resulting change in the value of the partial derivatives has hardly any impact on the simulation results. Therefore, following procedure has been implemented for the models of the CO<sub>2</sub> capture process.

Table 1: Simulation results of model with flash vessel and two control valves simulated for 2000 seconds.

Case	Valves	Vessel	Vessel	Total time [s]
	AllProps [s]	AllProps [s]	TwoPhaseDeriv [s]	
Reference (Valves normal "PT")	107.6	5.2	21.5	134.3
Valves "PT_1ph"	11.2	5.2	21.5	37.9
Vapour density transfer	7.2	5.2	21.5	33.9
Valves "PT_1ph", CallID	11.1	5.2	4.5	20.8

An additional variable (CallID) has been added to each two-phase component assigning each partial derivative function a unique identification. With this identification it is possible to distinguishably store simulation results in the Modelica-FluidProp interface of the different two-phase derivative calls throughout the modelled process (eg. LP Vessel CallID=1, MP Vessel CallID=2,...). Assuming the derivative function with the ID 1 is executed at a certain simulation time instance, then a check is performed if the difference between the current and the values from the previous time instance (which have been stored in the interface) are below a defined threshold. If that is the case, no property computation for the partial derivatives is performed but the stored results of the derivatives from the previous time instance are returned straight to Modelica. In case the threshold is exceeded, then a normal property calculation is performed with the external tool and the previous results stored in the interface are overwritten. This procedure has the benefit that computational time is saved if the change in the absolute value of the partial derivatives is marginal and thus has no impact on the solution. When modelling complex processes involving two-phase multi-component fluids a significant reduction in computational time can be obtained. Exemplary a comparison is provided for the model of the flash vessel with two control valves (see Table 1). The time spent for derivative computations reduces from 21.5 to 4.5 seconds for such a simple model.

## 5 Recommendations for a future interface

The presented model development approach leads to solvable models by making a smart choice for the thermodynamic and system state variables and by manually applying measures in order to keep the system in index-1 form. However, for reasons of numerical robustness, simulation speed and ease of initialization, a different choice of state variables might be more convenient than the one where the differentiated variables

are used as states. In order to allow for flexible state variable change and automatic index reduction partial derivatives of the thermodynamic properties are essential when using external tools, as demonstrated in this paper.

The goal is to design a Modelica library that interfaces to external property packages, whereby Modelica tools can automatically compute the total time derivatives of each variables in a set A (e.g. density, specific energy, specific enthalpy, ...) with respect to any meaningful subset of variables in a set B (e.g. pressure, temperature, specific enthalpy, ...) that uniquely identifies the thermodynamic properties of the fluid, including multi-component fluids and two-phase mixtures. This is required to successfully carry out the index reduction and/or state variable change task automatically. A first attempt on how to perform automated state variable change is presented by Wellner et al. [16].

At a higher level, this requires setting up a Modelica infrastructure where annotations point to the appropriate functions to compute all the required derivatives. At a lower level, it has to be ensured that the external property package can compute all required derivatives efficiently, i.e., by avoiding unnecessary duplicate computations.

## 6 Conclusions

This paper presents the development of a system model for a pre-combustion CO<sub>2</sub> capture process as part of an integrated gasification combined cycle power plant, which entails the modelling of highly non-ideal, two-phase multi-component mixtures. As this type of mixtures are currently not supported by available Modelica media libraries, an interface prototype was developed and tested with the fluid property package FluidProp for the modelling and simulation of the CO<sub>2</sub> capture process. Due to limitations regarding index reduction if using external property functions, an approach on how to develop index-1 models and choose system state variables as well as thermodynamic states appropriately is discussed.

For this type of simulations it appears that the computation of thermodynamic properties, in particular phase equilibria, accounts for the main share of the simulation time, therefore various ideas (single versus two-phase property calculations, decrease in redundancy, approximation of partial derivatives) are presented targeting computational efficiency. Further developments might focus on facilitating automated index reduction and making a wide range of partial derivatives available in a flexible and efficient manner. In an ideal setting, the tool allows for different choices of state variables and different solution strategies for the DAE system to find the best combination for the specific case of study.

## Acknowledgements

The work documented in this paper has been performed within the CO<sub>2</sub> Catch-up R&D programme aimed at demonstrating and optimising pre-combustion CO<sub>2</sub> capture technology for the energy sector. This programme is executed in a consortium of Vattenfall, TU Delft and ECN.

## References

- [1] K. Damen, M. Troost, A. Faaij, W. Turkenburg, A comparison of electricity and hydrogen production systems with CO<sub>2</sub> capture and storage. part a: Review and selection of promising conversion and capture technologies, *Progress in Energy and Combustion Science* 32 (2) (2006) 215–246.
- [2] K. Damen, R. Gnutek, J. Kaptein, N. Nannan, B. Oyarzun, C. Trapp, P. Colonna, E. van Dijk, J. Gross, A. Bardow, Developments in the pre-combustion CO<sub>2</sub> capture pilot plant at the Buggenum IGCC, *Energy Procedia* 4 (2011) 874–881.
- [3] F. Casella, C. C. Richter, ExternalMedia: a Library for Easy Re-Use of External Fluid Property Code in Modelica, in: *Proceedings 6th International Modelica Conference*, 2008, pp. 157–161.
- [4] J. Gross, G. Sadowski, Perturbed-Chain SAFT: An Equation of State Based on a Perturbation Theory for Chain Molecules, *Ind. Eng. Chem. Res.* 40 (2001) 1244–1260.
- [5] N. R. Nannan, C. M. D. Servi, T. van der Stelt, P. Colonna, , A. Bardow, An equation of state based on pc-saft for physical solvents composed of polyethylene glycol dimethylethers, *Industrial and Engineering Chemistry Research* 52 (2013) 18401–18412.
- [6] P. Colonna, T. van der Stelt, A. Guardone, Fluid-Prop: a program for the estimation of thermo-physical properties of fluids. Version 2.4, software (2004-2012).  
URL <http://www.FluidProp.com>
- [7] F. Casella, A. Leva, Modelling of thermo-hydraulic power generation processes using modelica, *Math. Comput. Model. Dyn. Syst.* 12(1) (2006) 19–33.
- [8] F. Casella, A. Leva, Object-oriented modelling & simulation of power plants with modelica, pp. 7597–7602, in *Proceedings 44th IEEE Conference on Decision and Control and European Control Conference 2005*, Seville, Spain, Dec. 12-15, 2005.
- [9] J. D. Seader, E. J. Henley, *Separation Process Principles*, Wiley, 1998.
- [10] J. Stichlmair, J. L. Bravo, J. Fair, General model for prediction of pressure drop and capacity of countercurrent gas/liquid packed columns, *Gas Separation & Purification* 3 (1989) 19–28.
- [11] R. Billet, M. Schultes, Prediction of mass transfer columns with dumped and arranged packings: Updated summary of the calculation method of Billet and Schultes, *Chemical Engineering Research and Design* 77, Issue 6 (1999) 498–504.
- [12] Infochem Computer Services Ltd, Multiflash - Flow Assurance Software, [www.kbc.com](http://www.kbc.com) (2014).
- [13] Richard Szczepanski, Infochem Computer Services Ltd, Private communication (2014).
- [14] F. Casella, J. Van Putten, P. Colonna, Dynamic simulation of a biomass-fired steam power plant: a comparison between causal and a-causal modular modeling, *ASME International Mechanical Engineering Congress and Exposition*, *Proceedings* 6 (2008) 205–216.
- [15] M. L. Michelsen, J. M. Mollerup, *Thermodynamic Models: Fundamentals & computational aspects*, second edition Edition, Tie-Line Publications, 2007.
- [16] K. Wellner, C. Trapp, G. Schmitz, F. Casella, *Interfacing Models for Thermal Separation Processes with Fluid Property Data from External Sources*, submitted to the 10th Modelica Conference, Lund, Sweden (March 10-12, 2014).



# Dynamic modelling of a parabolic trough solar power plant

Robert Österholm<sup>a</sup>, Jens Pålsson<sup>b</sup>

<sup>a</sup>Lund University, LTH, Department of Energy Sciences, Lund, Sweden

<sup>b</sup>Modelon AB, Ideon Science Park, Lund, Sweden

[osterholm.r@gmail.com](mailto:osterholm.r@gmail.com), [jens.palsson@modelon.com](mailto:jens.palsson@modelon.com)

## Abstract

Models for dynamic simulation of a parabolic trough concentrating solar power (CSP) plant were developed in Modelica for the simulation software tool Dymola. The parabolic trough power plant has a two-tank indirect thermal storage with solar salt for the ability to dispatch electric power during hours when little or no solar irradiation is present. The complete system consists of models for incoming solar irradiation, a parabolic trough collector field, thermal storage and a simplified Rankine cycle. In this work, a parabolic trough power plant named Andasol located in Aldeire y La Calahorra, Spain is chosen as a reference system. The system model is later compared against performance data from this reference system in order to verify model implementation. Test cases with variation in solar insolation reflecting different seasons is set up and simulated. The tests show that the system model works as expected but lack some of the dynamics present in a real thermal power plant. This is due to the use of a simplified Rankine cycle. The collector and solar models are also verified against literature regarding performance and show good agreement.

*Keywords: concentrating solar power; parabolic trough; solar salt thermal storage; dynamic modeling; Dymola; Modelica.*

## 1 Introduction

Thermal solar power is a technology of which the solar irradiation on earth is harvested in order to produce power in the range of village type application of several kilowatts all the way up to grid connected power plants producing several hundred Megawatts. It is an environmentally friendly way of producing power not contributing to the rising amount of greenhouse gases and other hazardous particles that

are let out in the atmosphere by traditional fossil fired power plants.

The principle is to concentrate the solar irradiation using a configuration of mirrors to produce high temperature heat. The heat is then transported to a boiler where steam is produced for a steam turbine or the heat can be used to power a heat engine. A generator coupled with the turbine or heat engine is then producing electrical power. Some systems also have the capability to store heat for use during for instance cloudy days and at night. There is a range of different configurations available both in the way solar irradiation is concentrated and how the produced heat is used to generate power. The most common, in regards of development and number of units built, thermal solar power plants will be presented.

A thermal solar power plant can be divided into three subsystems consisting of the collector, the thermal storage and the power cycle. Another important aspect of the system is the medium that is used to transport heat from the collector field to the power cycle and the medium used for thermal storage. The different characteristics and the most common medium used in a typical thermal concentrating solar power plant will be presented.

### 1.1 Collector

The concentrating device of a thermal solar power plant consists of mirrors, the collector, focusing the incoming solar irradiation on to a heat-absorbing device, the receiver. This is done because the solar irradiation per square meter on earth is too small to heat anything to a desired temperature used in power generating applications. By concentrating the irradiation from a large area on to a small point high temperatures can be reached [1]. The arrangement of the mirrors differs depending on which type of CSP plant configuration that is used.

Most mirror configurations use a tracking system to follow the movement of the sun in the sky in order to maximize the heat collection throughout the day. This can be done with either one (east-west direction) or two axes (additionally north-south direction). Systems using one axis require less investment and maintenance cost at somewhat lower performance compared to two-axis tracking system.

### 1.2 Thermal storage

Thermal storage capability is the key to cost efficient and flexible CSP plant operation. With thermal storage the plant owner is able to dispatch power when it's most valuable which often is in the evening when there is low or little incoming solar irradiation and the electricity price is high. There are a number of thermal storage solutions that are both in operation and under development. The type that has gained the most attention in recent years is a system where heat is stored in molten nitrate. Other types use rock, sand and oil as storage media [2].

### 1.3 Power cycle

Most currently operating solar trough and power tower systems use a Rankine cycle for electricity production. Water is heated up in a boiler producing high-pressure steam that is fed into a steam turbine coupled to a generator. Temperature and pressure data depends on the turbine design and heat source. In CSP plant applications the thermal fluid medium reaches a temperature of 400 to 600 °C. This means that steam temperature of around 350 to 550 °C and a pressure of 100 bar is common. Some CSP plants use a heat engine (such as Stirling motor or Brayton cycle) to produce power. Most solar plants are located in sunny and dry climate and use an air cooled condenser instead of the more common use of water as cooling medium [3].

### 1.4 Thermal media

Different heat absorbing fluids are used in CSP plants operated today. The fluid is often mentioned the thermal heating fluid or just "THF" and is used to transport heat from the concentrating apparatus to the power cycle. Some systems that use a heat engine don't use any thermal heating fluid. The most common type of THF is a thermal oil that can be heated up to around 400 °C. There are also systems that use molten salt mixture which can be heated to around 500 °C, but need an advanced control system to prevent the salt to crystallize at around 200 °C. [4] [5].

## 2 Parabolic trough and other concentrating solar power technologies

In the parabolic trough type CSP plant the solar irradiation is concentrated by parabolic curved, trough-shaped reflectors onto a receiver pipe running along the inside of the curved surface. The medium in the receiver pipes is heated by the concentrated solar energy and used to produce steam in a Rankine cycle. The mirrors are typically placed in parallel rows along a north to south axis to be aligned towards the sun. A tracking system angles the mirrors to minimize the incidence angle from the solar beams. Trough systems can incorporate thermal storage. Many currently operating systems are hybrids meaning they can be supplementary fired with an alternative fuel (often natural gas) when the solar irradiation and heat storage is not enough to run the plant. A typical parabolic trough plant can produce 50 MW to 100 MW of electrical power. One of the largest operating CSP plants today, the Shams 1, is of parabolic trough type. It has a power output of 100 MW and is located in the Abu Dhabi Emirate. [4] [5]

The linear Fresnel system works in the same way as the parabolic trough with another collector and mirror design. The absorber is elevated with several mirrors mounted under the absorber tube. A separate tracking device gives the mirrors high flexibility to track the movement of the sun. This system has a lower investment and operation cost than the parabolic trough system. The major disadvantage is the lower solar to electricity efficiency [8].

Power tower systems use one central receiver mounted on top of a tower. A large amount mirrors, so called heliostats, are placed around the tower and track the sun in two axes. The THF is heated to rather high temperature and is used to produce steam in a Rankine cycle. Recently molten nitrate has been introduced which is also the most common medium used in the thermal storage, eliminating the need for a heat exchanger between the thermal heating fluid and the thermal storage fluid [6].

The parabolic dish system utilizes a large two axis parabolic mirror concentrating the incoming irradiation on to a receiver in the mirror focal point. The receiver is typically a heat powered motor meaning that the absorbed heat is used directly in the receiver to produce electricity. The setup has been realized in plants producing up to 5 MW of electricity.

### 3 Reference system

The most common and mature technology for thermal solar power is the parabolic trough design with indirect two-tank thermal storage capability [9], and chosen as the subject of this study. In order to model the system correctly regarding incoming solar irradiation a specific location on earth is needed. The Andasol power plant in Aldeire y La Calahorra, Spain, was chosen as reference system because the availability of performance data and plant design parameters. Technical specifications of the Andasol power plant are presented in Table 1 below [11].

Plant name	Andasol-I and Andasol-II
Plant location	Aldeire y La Calahorra, Spain
Plant type	Parabolic trough
Start date	June 1, 2009
Receiver type	Schott PRT-70
Sun tracking	One axis in north-south direction
Collector type	Flabeg RP-3
Thermal heating fluid type	Dowtherm A
Turbine type	Siemens SST-700 50MW steam turbine
Thermal heat storage type	Two-tank indirect with molten solar salt

Table 1 Andasol power plant specification

The thermal heating fluid used in Andasol-I and Andasol-II are of the type Dowtherm A. The system modeled will use another type of THF manufactured by Eastman Chemical Company with product name Therminol VP-1. Therminol is commonly used in concentrating solar power plants today [12]. Also, more technical data is available about the Therminol heat fluid than the Dowtherm type.

### 4 Modeling the solar power plant

The different system components described in section one will be implemented in Dymola. There are no solar power specific components present in available Modelica libraries today. After model implementation of the components an evaluation will be made against available data. Furthermore, when the different components are working properly they will be connected to a system model that will be compared against performance data given by the operator of the reference system.

#### 4.1 Media models

Both the THF circulated through the receivers and the fluid used in the thermal heat storage need to be implemented in Modelica for use in Dymola. A sim-

plified temperature dependent medium model is developed where thermodynamic state is interpolated from a table. The THF called Therminol is a clear, transparent synthetic oil made of an eutectic mixture of 73.5 % diphenyl oxide and 26.5 % biphenyl. The oil has one of the highest thermal stabilities of all organic heat transfer fluids and is stable in liquid phase between 12 °C and 400 °C. This makes it ideal for use in CSP applications [13].

The fluid used in the thermal heating storage is a mixture of 60 %  $NaNO_3$  and 40 %  $KNO_3$  by weight often referred to as solar salt. Solar salt is used because of its thermodynamic properties such as high heat capacity, high density, relatively low cost and low vapor pressure. The low vapor pressure makes the salt storage easy because no pressurized tanks are needed. Another favorable feature of solar salt is the high temperature stability and liquid phase up to 560 °C. The downside is that it has a rather high melting point starting to crystallize at 238 °C. This means that special care must be taken to prevent the salt from freezing causing major damage to pipes, pumps and heat exchangers [14]. The thermodynamic properties of solar salt are implemented in the same way as the THF (Therminol).

#### 4.2 Sun model

A set of different equations is needed to calculate the direct solar irradiation onto a surface, in this case a solar collector assembly, for a specific date, time and location on earth.

The term solar time is used when calculating the angles that are needed to determine how much direct radiation will hit a collector, and differs from the local time on earth. The difference in minutes can be expressed as follows [15]:

$$\text{solar time} - \text{local time} = 4 \cdot (L_{st} - L_{loc}) + E \quad (1)$$

Where  $L_{st}$  is the standard meridian for the local time zone,  $L_{loc}$  is the longitude of the location in degrees west and E is the equation of time.

Equation of time is expressed as:

$$E = 229.2 \cdot (0.000075 + 0.001868 \cdot \cos B - 0.032077 \cdot \sin B - 0.014615 \cdot \cos 2B - 0.04089 \cdot \sin 2B) \quad (2)$$

Where

$$B = (n - 1) \cdot \frac{360}{365} \quad (3)$$

And n = day of the year [15].

The angle of incidence of direct solar radiation onto a north to south axis tracking collector that is used at the Andasol plant can be calculated as follows [15]:

$$\cos \theta = (\cos \theta_z^2 + \cos^2 \delta^2 \cdot \sin \omega^2)^{0.5} \quad (4)$$

Where  $\delta$  is the declination, the angular position of the sun at solar noon [15].

$$\delta = 23.45 \cdot \sin\left(360 \frac{284+n}{365}\right) \quad (5)$$

$\omega$  is the hour angle, the angular displacement of the sun east or west of the local meridian due to rotation of the earth on its axis at 15° per hour [16].

$$\omega = (t_{sol} - 12) \cdot 15 \quad (6)$$

$\theta_z$  is the zenith angle, the angle of incidence of beam radiation on a horizontal surface [15].

$$\theta_z = \cos \phi \cdot \cos \delta \cdot \cos \omega + \sin \phi \cdot \sin \delta \quad (7)$$

Where  $\phi$  is the latitude.

The incidence angle is the angular difference  $\theta$  between the normal to the aperture and the actual solar irradiation that can be seen in Figure 1. When the direct beam radiation is not normal to the plane of the collector aperture there is a loss of direct beam radiation that scale with the cosines of the angle  $\theta$ .

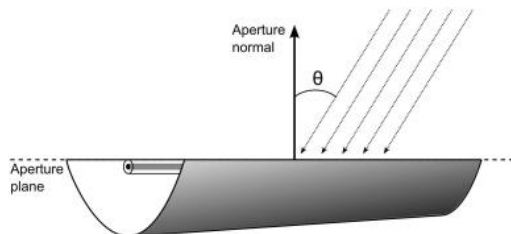


Figure 1 A graphical interpretation of the angle between the aperture normal and the solar irradiation [16].

### 4.3 Collector model

The collector is a mirror shaped as a half cylindrical parabolic reflector focusing the incoming solar irradiation onto the absorber pipe. There are numerous equations to describe the geometry of the mirror and the image, distribution of solar radiation flux across the focus, it produces. It is assumed that the collector design already is optimized which gives the parameters that should be provided to the collector model in Dymola. The parameters of interest are the optical efficiency and collector aperture width and length.

The aperture width is the trough width and the total aperture area is the length of the collector times the aperture width. The total aperture area gives the total amount of solar irradiation that can be concentrated onto the receiver.

There are several models for the mirror loss that have to be taken into account. In this model a single parameter called optical efficiency,  $\eta_{opt}$ , is introduced to describe the total mirror efficiency. The different losses included in this parameter are listed in Table 2 below [16].

Tracking error	Inability of the collector to perfectly orient along the tracking angle. Twisting of the collector about the lengthwise axis.	$\eta_{track}$
Geometry defects	Poor alignment of the mirror modules.	$\eta_{geo}$
Mirror soiling	Dirt or soiling on the reflective surface that decreases the reflected amount of solar irradiation onto the absorber pipe.	$\eta_{soil}$
General error	Any other loss not covered by previous categories.	$\eta_{gen}$

Table 2 List of mirror losses.

This gives the optical efficiency:

$$\eta_{opt} = \eta_{track} \cdot \eta_{geo} \cdot \eta_{soil} \cdot \eta_{gen} \quad (8)$$

This efficiency term is multiplied with the total radiation incidence on the collector calculated earlier in the sun model and the total aperture area which gives the amount of incoming, reflected solar irradiation onto the receiver.

$$\dot{Q}_{receiver} = I_{field} \cdot A_{tot,aperture} \cdot \eta_{opt} \quad [W] \quad (9)$$

### 4.4 Receiver model

The amount of heat absorbed by the THF in the receiver model considers the amount lost due to radiation and convection to the surroundings. The radiation and convection losses, either natural in case in no wind or forced in case of wind, are heat transferred from the outer glass envelope of the receiver. Because of the long receiver length, in the Andasol case almost 90000 m, the temperature and heat losses will vary along the absorber axis. In this case the model is discretized i.e. the absorber pipe is divided in n number of segments. The heat transfer equations are then solved in each segment summing up to the total heat transfer. By doing this a much more accurate model is obtained as discussed in several studies on the subject [17]. The receiver can be represented as a thermal resistant network as shown in Figure 3. The temperatures denoted in the equations that follow are marked in Figure 2 and Figure 3.

The absorbed energy in the receiver pipe is modeled with a standard conduction heat transfer model for a metal wall already available in Modelon Base Library (MBL). It is assumed that the pipe is made out of standard carbon steel. The heat transferred through the metal wall is then absorbed by the THF by convection in a pipe model also already available in MBL.

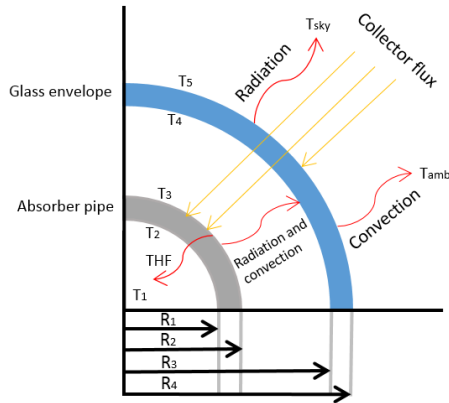


Figure 2 Measurements and heat flow in receiver.

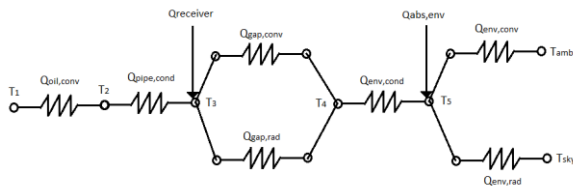


Figure 3 Receiver model represented as a thermal resistance network.

The outer surface in commercial absorbers is coated with special materials to decrease reflection from the surface. In the parameter settings for the receiver the surface emissivity of the pipe can be set to a value representing the current coating material.

From the pipe surface both convection and radiation occurs in the vacuum gap between the outer glass envelope and the pipe. The convection is modeled with a standard convective heat transfer model developed by Modelon. When this gap is under vacuum the convective heat transfer mechanism is molecular conduction between molecules. If the vacuum is lost for example by a crack in the glass cover the heat transfer mechanism is free convection [18]. It is assumed that vacuum is always present in the gap for this model and that the heat transfer coefficient for the vacuum of air is  $k_{c,gap} = 0.0001115 \text{ W/m}^2\text{K}$  [17]. The heat transfer coefficient is provided as a model parameter for the user which means that simulations with different vacuum properties or vacuum with different gases other than air can be made.

$$\dot{Q}_{gap,conv} = k_{c,gap} \cdot D_3 \cdot \pi \cdot L_{rec} \cdot (T_3 - T_4) \quad (10)$$

There were no models for radiation present in any Modelica library which had to be developed and based on following expressions:

$$\dot{Q}_{gap,rad} = \pi \cdot D_3 \cdot L_{rec} \cdot \gamma_{gap,rad} \cdot (T_3 - T_4) \quad (11)$$

$$\gamma_{gap,rad} = \sigma \cdot (T_3^2 - T_4^2) \cdot \frac{T_3 + T_4}{\frac{1}{\epsilon_3} + \frac{D_3}{D_4} \cdot \left(\frac{1}{\epsilon_4} - 1\right)} \quad (12)$$

Several assumptions were made in deriving this equation but the errors caused are relatively small [17]. The glass envelope is implemented as a heat transfer model including wall material with constant parameters developed by Modelon. The user needs to provide data for the glass material such as density, specific heat capacity and thermal conductivity. Most receiver glass envelopes are made out of borosilicate glass that has a very low coefficient of thermal expansion. This makes them resistant to thermal shock that is crucial in CSP applications [23].

Most of the solar irradiation concentrated onto the receiver is utilized by the THF but the receiver glass envelope also absorbs some heat. This raises the glass surface temperature and creates a temperature difference to the ambient. The amount of heat absorbed by the glass can be expressed as follows [19]:

$$\dot{Q}_{env,abs} = \dot{Q}_{receiver} \cdot \frac{\alpha_{env}}{\tau_{env} \cdot \alpha_{abs}} \quad (13)$$

Some of the heat transferred through the glass envelope is lost by radiation due to temperature differences between the glass and the sky. Convective heat transfer also occurs and is either natural or forced depending on if there is wind or not.

The radiation loss can be expressed as follows when assuming the glass envelope to be a small convex gray body surrounded by a large black body that represents the sky [17]:

$$\dot{Q}_{env,rad} = \sigma \cdot \pi \cdot D_4 \cdot L_{rec} \cdot (T_5^4 - T_{sky}^4) \quad (14)$$

The sky temperature can be assumed to be 8 degrees lower than the ambient temperature [20].

The convection from the outer surface of the glass envelope can be expressed as follows:

$$\dot{Q}_{env,conv} = h_{amb} \cdot \pi \cdot D_4 \cdot L_{rec} \cdot (T_5 - T_{amb}) \quad (15)$$

Where  $h_{amb}$  is the convective heat transfer coefficient to ambient.

For natural convection (no wind) the heat transfer function can be expressed as follows [17]:

$$h_{amb,nowind} = \overline{Nu} \cdot \frac{k_{56}}{D_4} \quad (16)$$

Where the Nusselt number can be calculated using a correlation developed by Churchill and Chu for a long isothermal horizontal cylinder [17].

All thermodynamic properties are calculated at the mean temperature between the envelope surface and the ambient temperature  $T_{56} = \frac{T_5 + T_{amb}}{2}$

Forced convection heat transfer will occur at windy conditions and is calculated in the same way as for natural convection but the Nusselt number will instead be estimated with Zhukauskas' correlation for external forced convection flow normal to an isothermal cylinder as follows from [17].

When the heat loss by radiation and convection from the envelope is known, and the same for the vacuum gap, the amount of heat absorbed by the thermal heating fluid is known:

$$\dot{Q}_{oil,conv} = \dot{Q}_{absorber} - \dot{Q}_{gap,rad} - \dot{Q}_{gap,conv} \quad (17)$$

$$\dot{Q}_{gap,rad} + \dot{Q}_{gap,conv} = \dot{Q}_{env,rad} + \dot{Q}_{env,conv} - \dot{Q}_{env,abs} \quad (18)$$

#### 4.5 Thermal storage model

The thermal storage task is to supply extra heat to the plant when the solar irradiation is below a certain value. In this case the outlet temperature of the THF from the collector field cannot reach the desired value of 393 °C, the design point temperature for the Andasol power plant. This is when the thermal storage instead is used heating the thermal heating fluid to 373 °C. It is not possible to reach the design point temperature because of losses in the heat exchanger between the thermal oil and the solar salt.

The collector field for the Andasol power plant is over-sized in order to be able to produce heat for both charging the thermal storage and the power cycle during the whole day and not only when the insolation is peaking in the middle of the day. With no thermal storage the collector field size could be reduced with approximately 40 %. After a couple of hours when the thermal storage is completely charged too much heat is absorbed just for the power cycle and a part of the collector field is intentionally defocused [20].

The heat from the THF is transferred to the solar salt via a heat exchanger with a 10 °C pinch point. When the solar salt is used to heat up the THF the system is reversed and the thermal fluid is heated up via the same heat exchanger. The THF will then reach an outlet temperature 10 °C below the solar salt inlet temperature. It is assumed that the thermal storage tanks are well insulated and no heat loss occurs.

The charging of the solar salt is made by tapping off a part of the heated THF from the collector field and redirecting this to the heat exchanger between the thermal oil and the solar salt. The mass flow rate of the thermal fluid through the heat exchanger is given by a simple energy balance and the fact that the minimum mass flow rate to the power cycle is known.

The discharging of heat from the thermal storage is a bit more complicated. The collector field is shut off via valves creating a new loop for the THF that runs through the thermal storage and then directly to the power cycle. This isolates the main THF pump creating the need for a secondary pump which is only used when discharging the thermal storage. The mass flow rate of this pump is calculated for the power cycle THF inlet temperature of 373 °C. The user can set the threshold value of incoming solar irradiation, the minimum amount of incoming solar irradiation where the system switches to run on the thermal storage. Under all simulations in this paper the value of 400 W/m<sup>2</sup> is used which means that the system will switch over to run on the thermal storage when the DNI is less than this value. The value has been chosen after testing the model and is not calculated.

#### 4.6 Rankine cycle model

The power cycle including boiler, steam turbine, generator and condenser is simply modeled. The Rankine power cycle model consists of a single heat exchanger where the THF flows on the primary side and the water boiled to steam on the secondary side. The water is supplied from a source and dumped in a sink meaning that there are no models for turbine, generator or condenser. By using this approach the thermal inertia of the otherwise big boiler is not accounted for. This means that the system modeled in Dymola will react too fast to changes in incoming thermal fluid temperature.

## 5 Modelica implementation

The Modelica language for dynamic simulations is ideal for CSP applications. The solar irradiation equations are time dependent and easily calculated after implementation in Modelica. There are different plant operating modes depending on solar insolation, amount of heat stored in the thermal storage and other events that has to be taken into account when the plant is simulated. There are also several modes the plant has to switch between in order to produce optimal amount of electricity to the grid. This kind of dynamics combined with automatic control, which Modelica is well suited for, makes it possible to test and simulate plant performance and behavior in a fine resolution. The downside with Modelica is a rather high threshold for new users.

The system model consists of Modelica implementation of the main components described and connecting them using system controller, see Figure 4.

- Sun model based on 15-minutes time series weather data on a file from a location close to Andasol plant in Spain
- Collector model consisting of own implementations of heat resistance and mirror models, and base classes (pipe and wall models) from Liquid Cooling Library (LCL) and Thermal Power Library (TPL)
- Thermal heat storage model implemented with own models combined with efficiency based heat exchanger model from MBL
- Simplified Rankine cycle solely consisting of source and sink terms and the same base class heat exchanger model from MBL
- Additionally a pump like component from LCL (setFlowRate) and an expansion volume component from MBL used for the thermal oil circuit.
- Thermal heating fluid (Therminol) implemented as table based media template class from LCL
- System controller for controlling mainly thermal heat storage as well as flow rate in the thermal oil and Rankine cycle circuits
- A summary record for key parameters and results

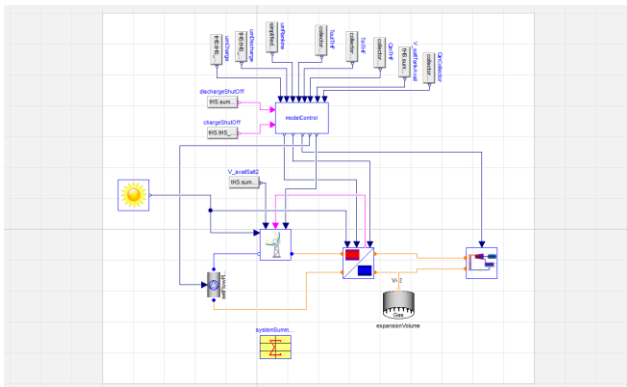


Figure 4 System model as in the Dymola Modeling view

## 6 Verification of system model

To verify the system performance a comparison with the Andasol I parabolic trough power plant is carried out with the same design parameters provided by the plant operator [22]. The goal is to match efficiencies from different parts of the system with them from the Andasol power plant. The thermal storage will not be used in this test, neither for heating up the salt nor to heat up the thermal fluid. They are left out from the simulation due to efficiency calculations that will turn out wrong if heat is added or subtracted from the

system. The performance numbers given by Andasol are based on an average typical weather year. To match the incoming solar irradiation a median value of the yearly incoming solar irradiation for a typical meteorological year were used as input to the system model and the specific simulation day was set to day no. 92 and the time to 10 AM. This represents a median value of the solar position.

Efficiency	Andasol	System model
Solar field – solar irradiance to steam	43 %	42.6 %
Rankine – steam to electricity	38.8 %	38.8 %
System – solar irradiance to electricity	16 %	16.5 %

Table 3 Results from system model validation.

As shown in Table 3 the values from the system model align well with plant data indicating a correct implementation of the system model. However, the Rankine efficiency is the whole cycle efficiency but in the system model only a simplified Rankine cycle is used. Therefore the same efficiency provided by Andasol was used to calculate the amount of electricity produced by the system model.

## 7 Results from test cases

In this section two different simulations from a clear and a partly clouded summer day respectively will be shown. By studying the behavior of the system model under these conditions, understanding of how the system handles solar irradiation variations can be gained. Information about system performance during a single day and how the system handles low values of solar insolation can also be obtained.

### 7.1 Simulation of a typical clear summer day

Flabeg, the manufacturer of the solar collector assemblies for the Andasol power plant is demonstrating the performance of the plant for a typical clear summer day in Figure 5. This figure serves as a benchmark and gives a hint of how the system model in Dymola should behave under similar conditions. The system model is set up to run on day 189 i.e. 8th of July, with the simulation starting at 4 AM and running for 24 hours. The results are presented in Figure 6 below.

As can be seen the simulation in Dymola is a bit more irregular than the presented graph from Flabeg, for instance some very sharp curves. The most obvious reason for this is that the graph of Figure 5 is a computer rendering and not based on actual calculations, which is the case for the other graphs.

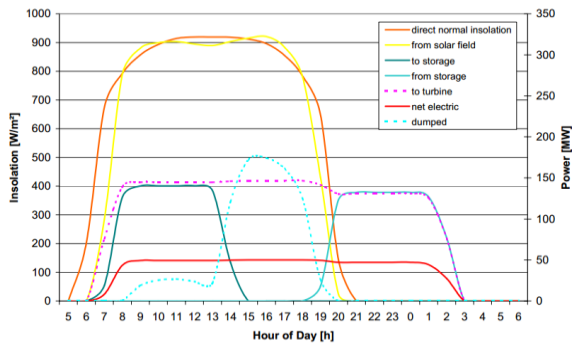


Figure 5 Insolation and heat flow to and from different system parts of the Andasol power plant [21].

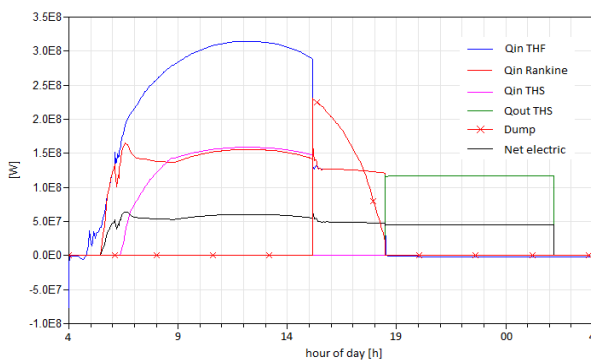


Figure 6 Heat flow to and from different system parts in the system model in Dymola.

There is a problem with the automatic control circuit causing the fluctuations in the beginning of the simulation in Figure 6. This could be avoided if the control parameters were to be optimized.

### 7.2 Partly clouded typical summer day

How the system behaves during a day which is partly clouded is investigated next with a custom set of solar data parsed to the system based on the data from the typical clear summer day, day no. 189. The incoming solar irradiation during the simulation is shown in Figure 7. The sudden drop in incoming solar irradiation is supposed to mimic a big cloud moving over the solar power plant. After three hours the cloud has passed and the solar irradiation goes back up again until the end of the day. The system results are shown in Figure 8.

At first the system behaves as in the clear summer day case shown in Figure 6 but when the sun is blocked by the cloud the thermal storage starts to discharge heat and the power plant can continue deliver power to the grid.

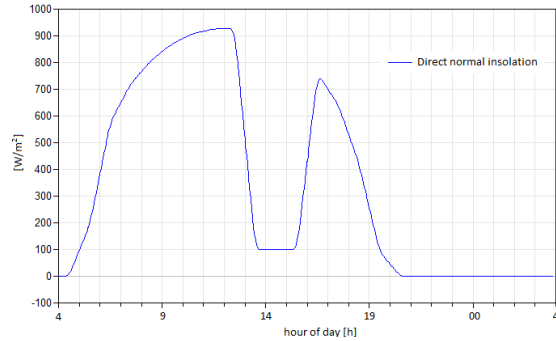


Figure 7 Direct normal insolation for the partly clouded day simulation.

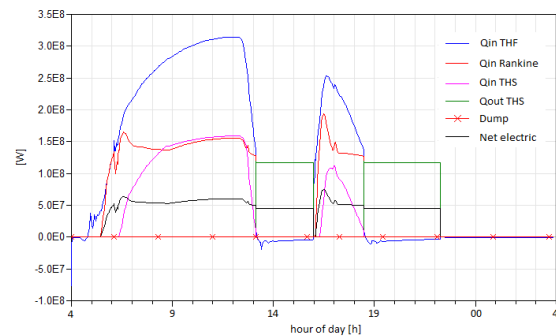


Figure 8 Heat flow from and to different system parts for the partly clouded day simulation in Dymola.

When the cloud passes three hours later the system switches back to run on heated thermal oil from the collector field until the sun sets. Because of the blocking of the sun the thermal storage is not fully charged when the irradiation disappears and the system can only run on the thermal storage for just little under four hours. Another control strategy may be desirable in this case depending on the price fluctuation of electricity. If the price is higher in the evening, which it usually is, the system should not start to discharge heat from the thermal storage during the day. Instead the power cycle could be shut off or put at low load during these hours. The power cycle could then go back up and deliver maximum electrical power in the evening while utilizing the energy contained in the thermal storage.

## 8 Discussion

The goal with this paper was to successfully model and simulate a concentrating solar power plant of a parabolic trough type and compare results with data found in literature. Parabolic trough is the most common technology with a huge potential to deliver large amount of environmental friendly electricity.



In order to model other types of CSP plants the current models may be redesigned rather easily for a linear Fresnel system. However, this is not the case for the central receiver solar tower design and the parabolic dish type.

The simulation results (numbers) presented may not be very useful in its current form but the models developed can be used for research and future plant analysis and development. There is some old code already existing for dynamic simulations of parabolic trough plants in other simulation tools. These tools do not offer as high degree of flexibility as the models developed for Dymola. The Modelica models can easily be altered and customized for a specific plant design and shortens the project lead time in the design and verification project phases where simulation is important. If systems can be modeled with a high degree of accuracy the time from project start to plant startup can be shortened and the plant performance predicted on an early stage. New system design can be tested and evaluated in an early stage in Dymola and less physical prototype tests have to be done which is much more cost effective and leads to cheaper plant development cost.

The verification done on the solar and collector models and on the system model as a whole, shows good results, which is an indicator that a correct implementation of the loss models has been made. The collector and sun models are considered to be implemented adequately but the thermal storage model needs more work because of the simplifications.

No complete Rankine cycle was implemented which is a drawback but the overall results presented show strong similarities with manufacturer data.

The fast switching between modes, charging and discharging of the thermal storage, and certain irregular behavior could be avoided with better automatic control with parameters optimized for each simulation depending on how much solar irradiation hits the collector during the day. The effects of this dynamics have to be studied more carefully to gain knowledge of which control parameters to modify.

## 9 References

- [1] P. Menna och D. Rossetti, "Concentrating solar power - from research to implementation," Office for official Publications of the European Communities, Luxembourg, 2007.
- [2] W. Stine och M. Geyer, "Chapter 11 - Energy storage," 29 10 2013. [Online]. Available: <http://www.powerfromthesun.net/Book/chapter11/chapter11.html>.
- [3] W. Stine och M. Geyer, "Chapter 12 - Power Cycles for Electricity Generation," 29 10 2013. [Online]. Available: <http://www.powerfromthesun.net/Book/chapter12/chapter12.html>.
- [4] "Shams 1 factsheet," 29 10 2013. [Online]. Available: <http://www.shamspower.ae/resources/media/Factsheet-ShamsFlyerEnglish.pdf>.
- [5] "Technology Characterization Solar Parabolic Trough," 29 10 2013. [Online]. Available: [http://www.solarpaces.org/CSP\\_Technology/docs/solar\\_trough.pdf](http://www.solarpaces.org/CSP_Technology/docs/solar_trough.pdf).
- [6] "Technology Characterization Solar Power Towers," 29 10 2013. [Online]. Available: [http://www.solarpaces.org/CSP\\_Technology/docs/solar\\_tower.pdf](http://www.solarpaces.org/CSP_Technology/docs/solar_tower.pdf).
- [7] "Technology Characterization Solar Dish Systems," 29 10 2013. [Online]. Available: [http://www.solarpaces.org/CSP\\_Technology/docs/solar\\_dish.pdf](http://www.solarpaces.org/CSP_Technology/docs/solar_dish.pdf).
- [8] M. Gunther, "Linear Fresnel Technology," i *Advanced CSP Teaching Materials*, Enermena;Deutsches Zentrum fur Luft- und Raumfahrt.
- [9] "Concentrating Solar Power Projects by Country," 29 10 2013. [Online]. Available: [http://www.nrel.gov/csp/solarpaces/by\\_country.cf](http://www.nrel.gov/csp/solarpaces/by_country.cf).
- [10] "Gemasolar plant description," 06 06 2010. [Online]. Available: <http://www.torresolenergy.com/TORRESOL/gemasolar-plant/en>. [Använd 29 10 2013].
- [11] "Andasol-2," 29 10 2013. [Online]. Available: [http://www.nrel.gov/csp/solarpaces/project\\_detail.cfm/projectID=4](http://www.nrel.gov/csp/solarpaces/project_detail.cfm/projectID=4).
- [12] "Parabolic Trough Projects," 29 10 2013. [Online]. Available: [http://www.nrel.gov/csp/solarpaces/parabolic\\_trough.cfm](http://www.nrel.gov/csp/solarpaces/parabolic_trough.cfm).
- [13] Eastman Chemical Company, "Therminol VP-1 product description," 28 10 2013. [Online]. Available: <http://www.therminol.com/pages/products/vp-1.asp>.
- [14] T. Bauer och N. Breidenbach, "Overview of molten salt storage systems and material development for solar thermal power plants," DLR, Köln,Germany.
- [15] J. A. Duffie and W. A. Beckman, *Solar engineering of thermal processes*, Second

- Edition, Madison, Wisconsin: John Wiley & Sons, 1991.
- [16] J. M. Wagner och P. Gilan, "Technical manual for the SAM physical trough model," National Renewable Energy Laboratory, Colorado, 2011.
- [17] R. Forristall, "Heat transfer analysis and modeling of a parabolic trough solar receiver implemented in engineering equation solver," National Renewable Energy Laboratory - U.S. Department of Energy, Colorado, 2003.
- [18] Y. Xiong, Y. Wu, C. Ma, K. Traore och Y. Zhang, "Numerical investigation of thermal performance of heat loss of parabolic trough receiver," *Science China*, pp. 444-452, 5 February 2010.
- [19] F. Burkholder och C. Kutscher, "Heat loss testing of Schott's 2008 PTR70 parabolic trough receiver," National Renewable Energy Laboratory, Golden, Colorado, 2009.
- [20] R. Kistner, Interviewee, *Dumping of heat in the Andasol power plant*. [Intervju]. 18 September 2013.
- [21] U. Herrmann, M. Geyer och R. Kistner, "The Andasol Project - workshop on thermal storage for trough power systems," FLABEG Solar International, Germany, 2002.
- [22] A. Cobra, "Andasol 1 Thermo solar energy project information," [Online]. Available: [www.estelasolar.eu/fileadmin/ESTELAdocs/document/powerplants/Andasol.pdf](http://www.estelasolar.eu/fileadmin/ESTELAdocs/document/powerplants/Andasol.pdf). [Använd 30 October 2013]
- [23] Cambridge Glassblowing Company "Borosilicate glass properties" [Online]. Available: <http://www.camglassblowing.co.uk/gproperties.htm>. [Used 30 October 2013]

# Testing Power Plant Control Systems in Modelica

Kilian Link<sup>a</sup>, Leo Gall<sup>b</sup>

Julien Bonifay<sup>a</sup>, Matthias Buggert<sup>a</sup>

<sup>a</sup>Siemens AG, Energy Sector, Erlangen, Germany

<sup>b</sup>Bausch-Gall GmbH, Munich, Germany

## Abstract

This paper describes the status and findings of the development of a Modelica based test system for power plant controls. The development is part of the ITEA2 project MODRIO which has the main goal to improve the operation of plants by utilizing modeling, simulation and optimization techniques.

The control test application shown in this paper demonstrates the usability of Modelica to run comprehensive tests for plant controllers involving large parts of the physical plant and the control system. However it highlights the need for further development, in areas such as: the test system, the discrete part of the models, the Modelica language, Modelica Association standards such as FMI and ModelicaXML and the tool support for these.

*Keywords: SCADA; DCS; ModelicaXML; MODRIO; Controller Test*

## 1 Introduction

Flexibility in operation of power plants becomes more and more important. This trend started in the nineties of the last century due to the opening of the energy markets world wide. The growing part of renewable energy production and their positive discrimination to be dispatched accelerated the evolution of the remaining fossil power plants. Therefore demand for control quality of these plants increased.

Power plants are controlled by distributed control systems (DCS) [6]. This article focuses on the Siemens DCS SPPA-T3000, see [3] for more details. However, most of the results are transferable to all up to date DCSs.

The goal of the development presented here is to allow the easy testing of controllers for power plants developed in a DCS such as SPPA-T3000 at every

control engineer's desk. Therefore the effort to run these tests needs to be acceptable and iterative improvement of the controls needs to be supported. Having such a tool available offers some great benefits:

- Tests on the real plant shall be avoided, since these are:
  - Expensive, due to cost of fuel and loss of profit.
  - Dangerous, since they might cause damages.
  - Limited with respect to the number of tests and the allowed operating range.
- Speeds up the development of improved controllers.
- Will improve the quality and enable systematic quality assurance (QA).

During the requirements engineering phase it became apparent that users, i.e. the control engineers, consider using a test system for very different tasks and in all different phases of a project. This means that the system under development needs to cover different aspects such as early test of concepts on the one hand and regression testing for maintenance of existing control standards on the other. Apparently, this needs high flexibility on the modeling and simulation for the plant model in order to cover all different kinds of physical effects of the plant in an appropriate level of detail.

An agile software development approach was chosen for this project, because the control engineers, modeling experts and software developers had the chance to team up for this development. Having a working prototype early on, turned out to be very useful.

Everything afore mentioned, lead in the direction of a fully Modelica based approach, at least for the fast development of a prototype. Due to the urgent need for control testing in a running project this fully Modelica based prototype has than be applied to a

real life example which highlighted some important advantages of the Modelica solution:

1. No special support of the DCS (besides already existing export features, see section 3).
2. Comprehensive insight in the control and plant model.
3. No real-time requirement.
4. No need to develop interfaces, e.g. to a virtualized control system such as SPPA-S3000 [2].
5. Only a Modelica tool needed for deployment.

The major disadvantage is that this approach introduces a new source for bugs: The Modelica model of the control system (see section 3). Compared to the advantages mentioned above this seems to be acceptable.

## 2 Background

### 2.1 SPPA-T3000

SPPA-T3000 is a Siemens DCS system of the latest generation. Like most of its competitors the communication back bone is the IEC Fieldbus standard. It offers integrated functionality for plant operation (including HMI), Alarming, Data Archives, Engineering of the controls, Diagnostics for optimized plant maintenance and interfaces for field device communication [3]. It is notable that the online runtime system of SPPA-T3000 still consists of programmable logic controllers (PLCs), i.e. the SIMATIC S7. But even so the power plant control needs to be run on a few of these PLCs for performance reasons, the DCS system still provides a global namespace for all signals.

The global namespace for all signals introduces a motivation for systematic naming. In power plants the KKS (Power Plant Classification System) system is still commonly used, although newer standards like RDP-PP based on EN 81346 are available. Typically the plant model is more general and therefore less strictly named. Disregarding which of the mentioned naming standards are used for the control system, the interface to the plant model needs to be taken into consideration for the design of a control test system for power plants.

### 2.2 SiemensPower

The Modelica library SiemensPower is used for dynamic simulations throughout the entire lifetime of a power plant: from conception to decommissioning.

The primary target is to provide information on dynamic behavior of a plant to verify the feasibility of process concepts and control strategies.

Validation of operating and safety procedures as well as tests of control concepts before implementation on real plants is performed, using SiemensPower models.

Key features of SiemensPower are:

- Wide range of component models especially for combined cycle power plants.
- Control package to implement real plant control in simulation models that allows parsing of controls from SPPA-T3000.
- High performance TTSE (Tabular Taylor Series Expansion) water/steam function.

In the context of this paper, all equations describing the dynamics of the plant and of controls are implemented in SiemensPower.

## 3 Bringing the Control System into Modelica

This section describes how functional block diagrams are automatically transferred from SPPA-T3000 to a controller model in Modelica which can be simulated together with the plant model. Figure 1 shows the two steps of exporting and converting functional block diagrams.

First, the control engineer has to define a set of functional block diagrams which are to be tested. This subset of the full-plant control system is then exported from SPPA-T3000 into an XML file describing the diagram layout, connections and parameter settings.

Second, the XML output is translated into a Modelica package containing one model for each functional block diagram and a top-level class containing all open controller inputs. Therefore a customized T3000-Modelica parser has been developed. During the conversion, the graphical layout of block diagrams is preserved as much as possible in order to allow analyzing and changing the control layout in the same style as in SPPA-T3000. But there are limitations for blocks with conditional inputs. For example, T3000 icons are automatically adapted in height, according to the number of activated input connectors. Unfortunately in current Modelica, the means of adapting icons and connectors according to parameter settings are very limited.

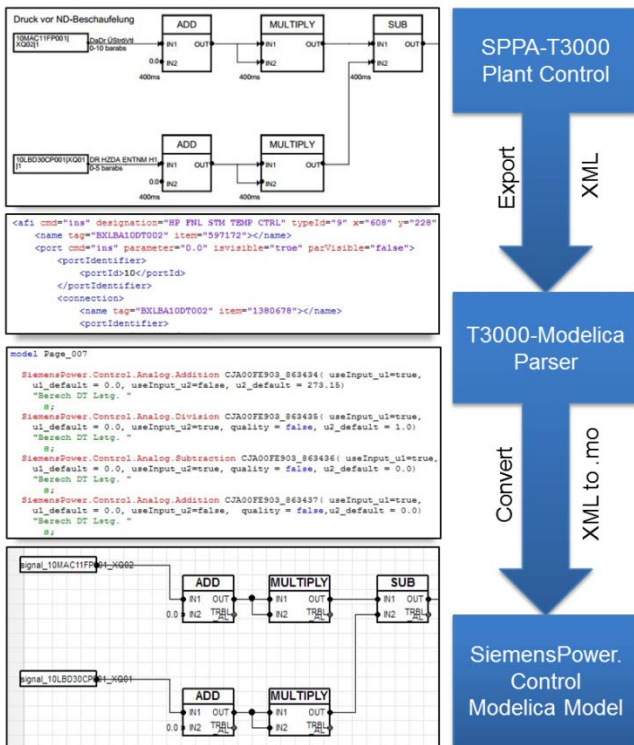


Figure 1: Process from SPPA-T3000 XML to Modelica

The parsed controller package contains the layout, hierarchy and parameterization of control blocks as Modelica instances, but lacks the actual equations. All relevant blocks have been re-implemented in Modelica, similar to the approach of [1]. There is an option to use purely discrete equations or continuous equations as much as possible. Re-implementing all controller equations and integrating them into the component library SiemensPower was a considerable effort. This work has already been initiated for earlier projects, as there was a need for real-world control blocks in dynamic simulation studies. The validation work of all control blocks is still ongoing.

The global name space of SPPA-T3000 (see section 2.1) is replicated by the connection to an outer expandable connector (bus). Therefore, the parsed functional block diagrams do not have input/output connectors on the top level, instead they are connected via bus signals in KKS naming scheme.

Due to extensive bus usage, the T3000-Modelica parser can avoid generating a lot of connect equations through the model hierarchy. Furthermore changes in the control layout are facilitated as all signals can be retrieved from bus at any level, just as in the authoring tool SPPA-T3000. Due to the high number of bus instances in the model, compiling was not possible. The reasons were memory and performance limitations during translation. This has been solved by using the inner/outer concept of Modelica.

## 4 Plant Model

The overall combined cycle power plant and its district heating system has been modeled based on the in-house Modelica library SiemensPower.

### 4.1 Description of Plant Model

For this prototype the control of a district heating system of a combined cycle plant (CCPP) was investigated. The district heating system uses steam from steam turbine extractions in order to heat the water that will be supplied for district heating. The main function of the controller is to control the supply temperature of the water by means of several valves in the plant. Figure 2 shows a simplified schema of the model.

The model of the CCPP serves as dynamic and consistent boundary conditions for the district heating system. It was therefore reduced to a minimal level of detail. The gas turbine is a steady state model based on tables, the steam turbine sections are modelled using Stodola’s equation and a constant isentropic efficiency and the heat recovery steam generator is a lumped time delay model as proposed by VDI 3508.

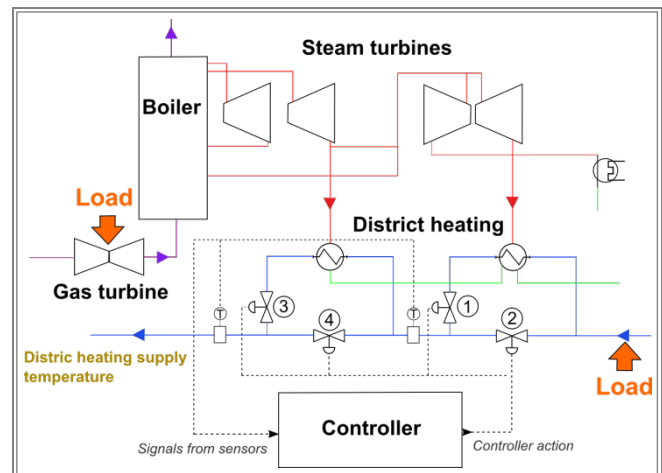


Figure 2: Simplified schema of the overall plant model with its controller

### 4.2 Components of District Heating System

A district heating system has long piping connections which can be longer than 100 meters, thus leading to large delays in the transport of enthalpy. To model this accurately while avoiding a huge number of elements in the discretized pipes, transmission line pipe models with spatial distribution for the enthalpy are used.

In the transmission line the enthalpy is delayed with a time  $\tau$ , solution of the following equation:

$$\text{Length of pipe} = \int_{t-\tau}^t \text{velocity}(t) dt$$

The controlled butterfly valves are modelled according to valve pressure loss correlation using a zeta (pressure loss coefficient) characteristic curve as function of opening angle:

$$\begin{aligned} \text{mass flow} &= \frac{\text{cross sectional area} \times \sqrt{2 \times \text{pressure difference} \times \text{density}}}{\text{zeta}(\text{angle})} \\ \text{massflow} &= \frac{\text{crosssectionalarea} \times \sqrt{2 \times \text{pressuredifference} \times \text{density}}}{\text{zeta}(\text{angle})} \\ \text{crosssectionalarea} &\times \frac{\sqrt{2 \times \text{pressuredifference} \times \text{density}}}{\text{zeta}(\text{angle})} \end{aligned}$$

To have realistic temperature transients, the heat exchange surfaces and especially the condensing heater need also to be correctly modeled. Hence, specific heat correlations for one and two phase flows are used [7].

Other important parts of the plant for a reliable controller test are the sensors. Due to the delay of enthalpy transport in the connection pipe, the exact position of the sensors is crucial. Further time delay within the sensors themselves must be also considered.

### 4.3 Size of Model

To improve the performance of the plant model an effort has been made to reduce the size of the non linear systems of equations at its maximum. This has been principally achieved by increasing the amount of continuous states, where appropriate. No system with higher amount of unknowns than 1 are left after manipulation of the non linear systems by the Modelica simulation tool (Dymola).

The plant model has around 464 continuous states and 4739 time-varying variables.

After adding the controller, the model keeps the same amount of continuous states as all controller variables are discrete and has in total 17095 time-varying variables.

## 5 Connecting Plant Model and Control System

A Test Unit is a Modelica model which instantiates one version of the plant model as well as the controller model (System under Test).

Figure 3 shows the current approach for signal handling. Besides connecting controller and plant model, the Test Unit is used for adding boundary conditions. Test Cases are then created by extending the Test Unit and adding disturbances and changing parameters. A benefit of storing all Test Cases in the Modelica package hierarchy is having one single source of information during simulation and testing.

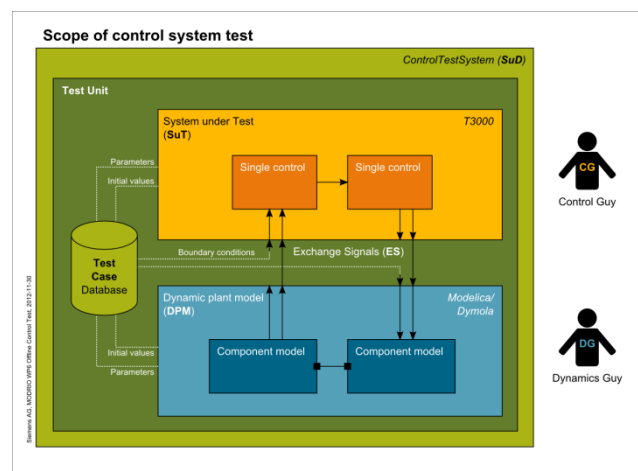


Figure 3: Test Unit with plant model and controller

The tested functional block diagrams usually depend on hundreds of Boolean and Real signals from the power plant control system. Therefore, setting appropriate boundary conditions is only possible in cooperation between control developers and dynamic simulation engineers. When connecting the models, additional unit conversions are necessary. The controllers work with non-SI units, which are adapted to different customer requests around the world. In order to avoid errors from manually converting units, an automated conversion of units is planned. The support through tools and the Modelica language for handling of non-SI units and factors could be improved (e.g. %).

Due to the chosen package structure, the controller can be updated (re-parsed) without requiring changes to the Test Unit, as long as no new boundary conditions are required. This enables efficient testing of various controller revisions.

## 6 Simulation

### 6.1 Performance Issues

The performance of plant model in open loop is very acceptable with a computation time considerably shorter than real time. Also, the computation of the discrete control signals at each one of its cycle time needs only an insignificant amount of computing resources. However, for the complete Test Unit, the time events generated at each cycle time of the control as well as the way how these time events are handled by the DAE solver dramatically deteriorate the performance of the plant model.

If it is not possible to avoid the generation of events at each control cycle time, one solution to improve the performance is by simply increasing the cycle time in order to reduce the amount of events. The cycle time in the DCS can typically go down to 100 ms. A cycle time of 400 ms can be chosen for the models, which greatly improves the performance. However, small instabilities in control actions can be noticed with this higher cycle time.

Another way to improve simulation performance is to change the DAE solver settings and the way it handles events. The Dassel solver starts for example its iteration after an event with the minimal step size, thus worsening the performance. While this can be modified for the Dassel solver in Dymola by manipulating the `dsin.txt` file, there is unfortunately a general lack of possibilities to influence solver settings in a user-friendly way.

An alternative attempt was to use FMI for Co-Simulation by packaging the controller in a FMU. This would offer the possibility to use two different solvers, a fixed step size solver (e.g. Euler) for the controller and a variable step size solver (e.g. Dassel) for the plant model. Due to the large size of the control model in terms of block instances (23000 in total) and amount of parameters, the simulation of the FMU in Dymola was unfortunately unsuccessful.

### 6.2 Results and Validation

A first validation test has been performed comparing the model behavior with measurement data from a real plant.

Boundaries of the model are set according to the measurements. As the model is covering the overall plant only few boundaries are needed. The necessary boundaries are the gas turbine load, the district heating load, and the set point for the district heating supply temperature.

The validation shows that the behavior of the model fits very well to the plant behavior. For instance, this can be observed for the valve openings on Figure 4.

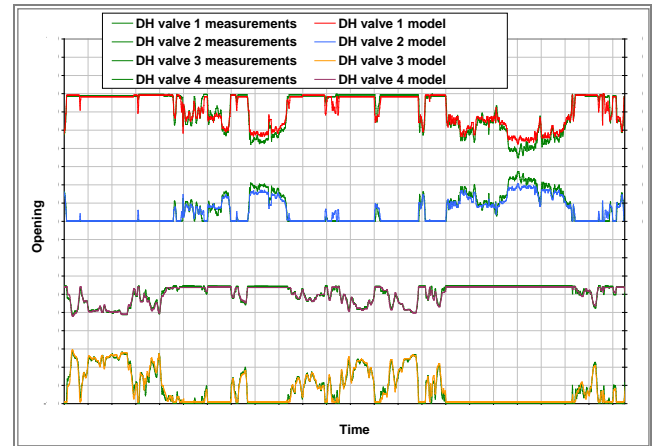


Figure 4: Comparison of real plant and model for the valve openings

## 7 Conclusion

This paper shows that the simulation based test of control systems in Modelica is possible for industrial use cases.

The self-contained simulation setup revealed interesting benefits. First of all, absolutely no changes to the authoring tool (SPPA-T3000) were necessary. Secondly, the tested output file of SPPA-T3000 could be the same as used on the real plant.

Even if the Functional Mock-Up Interface (FMI) is “en vogue” in industrial applications, the tool dependencies are much lower with our self-contained simulation model in Modelica. We can avoid a complex co-simulation setup for testing and are open to fast changes on the plant model and controller as well as Test Cases. All parts of the model retain their hierarchical structure and can be investigated deeply during testing. This helps control engineers not only to work with but also to trust in the simulation based testing procedure.

On the downside, we are reaching the limits of what current Modelica tools can handle in terms of number of signals and simulation performance. And of course there is substantial maintenance effort in regards to SiemensPower library and the T3000-Modelica parser in order to provide a stable control test system for in-house users.

## 8 Outlook

Encouraged by the success of the existing prototype the development of the control test system will be continued. Naturally the prototype needs to be improved with respect to many aspects and the most severe shall be mentioned here.

First of all, the controller part is a fully time discrete model which is modeled using existing Modelica elements such as “when” clauses and “pre()”. Besides the sampling rate or cycle time being defined for the global scope, i.e. contrary to the real DCS all controllers run on the same cycle time. This simplification did not limit the capabilities of the prototype for the particular case. However, it is a strong limitation in general. Note, the limitation is not on the export of SPPA-T3000 or the parsing to Modelica, but on the modeling of the functional controller blocks. It is evident that Modelica’s new synchronous features [4] will help to improve the situation. Both, on the modeling part utilizing clocks and sampled systems as well as on the simulation part guaranteeing deterministic behavior.

Another missing part to allow model based control development is the backwards parsing to SPPA-T3000. This step is needed to close the cycle in case modifications to the controller have been implemented during testing in Modelica. Again, the development of this feature might be backed up by a recent Modelica Association development, namely ModelicaXML. If this is standardized and supported by Modelica tools it might be a big step in the right direction. Then, the remaining task to transform one standardized XML format into another standardized XML format will be of very limited effort.

## Acknowledgements

The German Ministry BMBF has partially funded this work (BMBF funding code: 01IS12022A) within the ITEA2 project MODRIO [5].

## References

- [1] Marco Bonvini, Alberto Leva (2012). [A Modelica Library for Industrial Control Systems](#), Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany
- [2] Siemens AG, SPPA-S3000 Simulator, <http://www.energy.siemens.com/hq/en/automation/power-generation/simulation/sppa-s3000.htm>
- [3] Siemens AG, SPPA-T3000 Control System, <http://www.energy.siemens.com/hq/en/automation/power-generation/sppa-t3000.htm>
- [4] Hilding Elmqvist, Martin Otter, Sven Erik Mattsson (2012). [Fundamentals of Synchronous Control in Modelica](#), Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany
- [5] MODRIO - Model Driven Physical Systems Operation, <http://www.modrio.org>
- [6] Wikipedia, The Free Encyclopedia., Distributed control system, 19 January 2014, [http://en.wikipedia.org/wiki/Distributed\\_control\\_system](http://en.wikipedia.org/wiki/Distributed_control_system)
- [7] Verein Deutscher Ingenieure / VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen (GVC), VDI-Wärmeatlas, Edition 8, 1997



# Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis

Michael Tiller  
Xogeny Inc., USA  
michael.tiller@xogeny.com

## Abstract

Modelica has proven to be a compelling technology for creating sophisticated multi-domain models. It provides modern language features to promote model reuse and maximize developer productivity. These capabilities are backed up by proven simulation performance. More recently, the Functional Mockup Interface standard (FMI) has created an avenue for these models to be exported outside the model development environment as Functional Mockup Units (FMUs).

In this paper, we explore one possible way to utilize models that have been exported as FMUs. Specifically, we discuss how to incorporate these models into a web-based engineering analysis application that is designed to be accessible to non-expert users. Our goal is to show the important role that web and cloud based approaches can have in magnifying the impact of modeling activities across the enterprise.

We consider a specific engineering model and discuss exactly how we have transformed the model to make it accessible as a web-based application. This includes a discussion of the input and output data associated with the model as well as how a web based deployment (backed by cloud based services) can provide unique features compared to more conventional methods of model deployment.

**Keywords:** *FMI, cloud, web, HTML5, JavaScript*

## 1 Background

### 1.1 “Start the Revolution Without Me”

At the dawn of the world wide web, web servers served up static content (ordinary files) and web browsers rendered that content. What was remarkable about the web at this point was the ability to mix graphics and text using HTML and to hyperlink between pages. As the web matured, servers started to switch from serving static files to assembling content on a per request basis.

Up until 1998, the role of the web browser was simply to render this content produced by web servers. But in 1998, the “Web 2.0” revolution began. A big part of this revolution was the idea that the browser should become an application platform. Although at this point it wasn’t clear what technologies would ultimately win out for “executing” content through the web (Flash, Javascript or a variety of proprietary plugins), it was clear that a browser was no longer just for rendering hypertext.

Since 1998, the changes in the web browser platform have been nothing short of astounding. In fact, most users of these web browsers are probably not even aware of all the capabilities that have not only been added to web browsers but also standardized across browsers. Modern web browsers don’t just render hypertext, they can render vector graphics through SVG, detailed three dimensional scenes through WebGL, a wide variety of open ended rendering capabilities through canvas widgets and feature ever improving styling options through CSS. Along the way, the HTML specification has improved as well.

Behind the scenes, cloud based technologies are advancing by leaps and bounds. Every day new tools

and technologies emerge for supporting web-based applications. For example, there are now so many high-quality free options for databases that it is almost impossible to keep track of them. But this is true in nearly every area of web and cloud based technologies from Javascript frameworks to virtualization technologies.

The ubiquity of distributed computing resources is fundamentally changing the way we think about programming these large scale systems to make them more scalable and fault tolerant. This, in turn, has changed the way we think about programming. New languages and programming metaphors (*e.g.*, channels in Go[1][2], the `core.async` library in Clojure[3], actors in Akka[4] and promises[5]) are emerging to help us abstract away the behind the scenes complexity associated with these new distributed computing paradigms.

## 1.2 Engineering 2.0

But what does this mean for engineering? Almost nothing. While most engineers personal lives have been impacted somewhat by web based applications like Facebook, Twitter, Evernote, etc. their professional lives are still largely centered around applications, computing resources and storage tied to their local desktop. Simply put, the tools and capabilities in engineering are far from keeping pace with the proven technologies in other areas that have emerged over the last two decades.

Part of this effect is driven by the fact that engineering tools tend to be large and monolithic. This has resulted, in part, from industrial customers who approach adoption of engineering tools with a long list of requirements. This tends to foster a monolithic approach which, in an ironic twist, probably ends up hurting these customers in the long run.

One of the reasons that the technology industry seems to be better served by recent advances is that they are a more aggregated market. Engineering companies often attempt to differentiate themselves through what they feel are unique or innovative processes. The result is that their requirements are all slightly different from each other. This, in turn, means that vendors can rarely make a product that has broad market appeal in engineering. This leads to the situation that vendors often cannot justify sig-

nificant development resources or reinvestment because they are, ultimately, addressing a niche market. Moving forward, engineering and industrial companies need to collaborate more to drive standards and common best practices. In this way, more resources can be brought to bear on the problems that they all share.

While the web has evolved well beyond “web 2.0”, engineering is still waiting for the impact of those technologies to create an “engineering 2.0” revolution. Until then, engineering applications will remain largely centered around the desktop and subject to the same computing and storage limitations they always have and collaboration will be limited to “SharePoint” sites or network shared drives.

## 1.3 Xogeny

Xogeny was started to attack this problem head on. Xogeny is a new company with no legacy software to support. Everything we do starts with modern, proven technologies. There are countless technologies out there developed by technology giants like Google and Amazon that are simply there for the taking. Xogeny’s mission is to help build the bridges necessary to make these technologies accessible for engineering applications.

The first step in this process was to develop a platform to easily distribute simulation of FMUs to the cloud. We call this platform FMQ.

# 2 Representative Model

## 2.1 Vehicle Thermal Management

For the remainder of this paper, we will be discussing a Modelica model used to study vehicle thermal management. This model was developed by Modelon[6]<sup>1</sup> **without any prior consideration given to using it with the FMQ platform.**

The model itself is very sophisticated and includes detailed models of several different aspects of the vehicle. This model is used to simulate the thermal

---

<sup>1</sup>The author wish to thank Modelon for their participation in this case study and express his sincere appreciation for their support.

response of many individual components and subsystem during a prescribed drive cycle. Such analyses are important when optimizing system level efficiency to ensure that components are not oversized for their purpose and that control strategies are effective in managing thermal loads without reaching unacceptably high temperatures in components.

## 2.2 Model Compilation

The model itself was written in Modelica. As part of the normal modeling process (unrelated to any specific application involving FMQ), a considerable amount of engineering information was captured. For example, most parameters in this model include a description, physical units as well as minimum and maximum parameter values. Modelica makes associating such information with the model quite easy.

This Modelica code is then compiled into an Functional Mockup Unit (FMU) that conforms to the Functional Mockup Interface (FMI) specification[7]. One important thing to understand about this process of converting Modelica into an FMU is that much of the engineering information discussed previously is propagated into the FMU. So the descriptions, physical types and limits of parameters are available via the `modelDescription.xml` file in the FMU.

The FMI specification translates Modelica parameter values and solution variables into “flattened” names. This means that while the original Modelica model was hierarchical, the parameters and variables contained in the FMU are essentially just a simple list (not a tree structure) with fully qualified names that effectively indicate their location within the model instance hierarchy. The consequence of this is that organizational information from the Modelica model is only partially preserved (via the names).

## 2.3 Input and Output Data

As mentioned previously, variable names convey information about the relative positions of variables within the model hierarchy, but their organization within the FMU is flat. The parameters (input data) we are interested in characterize several different subsystems in the vehicle. These can be organized roughly into parameters related to the heat exchang-

ers, the powertrain and the chassis. Each of these different categories contains information about both the physical characteristics of components but also various control strategy calibration parameters as well. Also, some of these categories may have so many parameters associated with them that they necessitate further hierarchical organization. So, for example, the data associated with the heat exchangers is further broken down into information about stack geometry, fan control and scale factors.

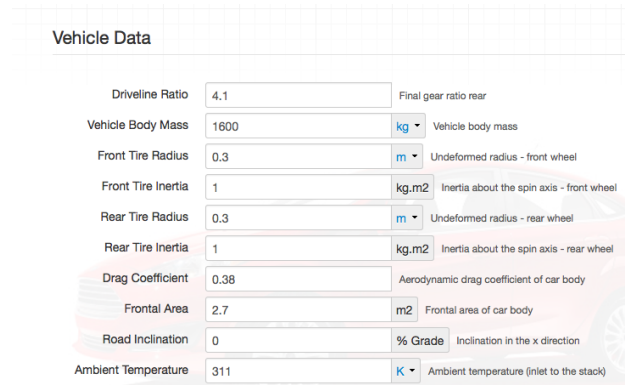


Figure 1: Input data panel

In addition to input data, there is also the question of what kinds of results can be generated from the FMU. In the case of this model, there are an enormous number of variables that are evaluated when the FMU is simulated. We won't attempt to list them all here, but the set of output variables consists of nearly any kind of information you might be interested in when exploring vehicle thermal management. As we will discuss shortly, we will only use a handful of these results. But it is worth noting that the FMQ platform itself is capable of computing and extracting whatever signals we need.

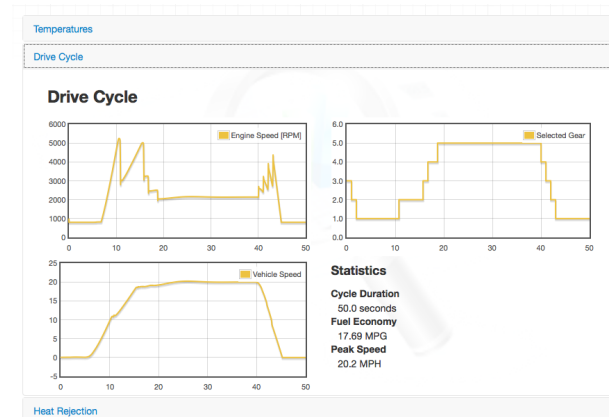


Figure 2: Output Report

### 3 Supporting Non-Experts

Before jumping into a discussion about web-based engineering analysis, it is worth taking some time to discuss **why** a web-based analysis tool is useful. The first thing to clarify is that this paper is not about web-based **development** tools. In other words, these tools are not designed for the creating Modelica models. The process of model development is a complex process and requires a sophisticated user interface. While creating a web-based development platform is a very interesting topic, it is not the topic of this paper. As such, for the purposes of this paper we assume that models are developed in an existing model development environment. The question we are focusing on is how to make these models accessible to non-developers or **non-experts in general**. So let us spend a little time considering a workflow that involves non-expert users.

Consider the following scenario. You are a software developer. You are writing software that you want lots of people to use. You work very hard developing the software and now you are ready to let other people use it. Imagine if circumstances required that instead of simply installing a compiled version of your code, users of your software had to buy all of your development tools (*e.g.*, Visual Studio), find a way to get the source code to your application, and then build your software for themselves.

If your “user” was an ordinary consumer (and not an open source hacker), this approach would be a complete disaster. Until recently if you were a Modelica developer, this would have been the only way to distribute your models **and** the compiler that your users would require could potentially be rather expensive. Fortunately, with the emergence of the FMI standard, we have an open and widely format for distributing models.

As a result, potential model “consumers” no longer have to compile the models themselves using expensive model development tools and their associated traps and pitfalls. However, simply having an FMU is not a complete solution for deploying models to non-expert users. There is still the issue of adding an appropriate graphical user interface. It is also important to recognize that non-expert users require an interface that is relatively straight-forward to use and warn users about potential mistakes. Such an application will typically expose a limited set of input and

output data. This data should be organized in a way that is intuitive to the user. Finally, an application developed for non-expert users must focus on addressing the specific questions of that user. If this means masking some or even most of the underlying models general capabilities, then so be it.

There is an (often uncaptured) return on investment for such efforts. Many model developers are forced to spend at least some of their time justifying the resources they consume for model development (in the form of time, expensive tools, etc). This concept of model deployment (*i.e.*, having an easy path for getting analyses leveraging your models into the hands of non-experts) is one potential way to demonstrate the need for model development resources. If model developers have an easy means to deploy models to end users and, as a result, turn those non-expert users into enthusiastic customers, then they have an opportunity to create a “pull” effect for their models. This can potentially lead to greater demand from the organizations that use those models. In such cases, this can lead to a virtuous cycle where the model developers can more easily justify their resource demands and, ideally, this increased demand will allow them to focus more resource on model development.

It is worth noting that non-expert users are generally interested in performing some kind of analysis. A model may be central to this analysis, but the analysis often involves more than just a single simulation of a single model. In a sense, a model is simply a complex function (*i.e.*, you give it data and it returns data). Creating and application and an high quality user experience involves much more than providing people with a function. We have deliberately used the term web-based engineering *analysis* applications because we think it is important to explicitly recognize that these applications must be prepared to support the complete analysis, not just part of it. Leveraging one or more models, a typical analysis will involve an optimization, a sensitivity study, a Monte-Carlo analysis or some other numerical procedure.

### 4 Web-Based Deployment

Given the requirements for non-expert users and the representative vehicle thermal management model previously discussed, the remaining question

is what should be the medium for model deployment.

## 4.1 Desktop Limitations

Given the case previously made for deploying models to non-expert users, one approach could be to develop desktop based analysis tools that are distributed to end users with the necessary models embedded in them. Indeed, one could argue that this is the conventional choice. But this approach has several important drawbacks.

The first issue to consider is data management. A tool that runs on a desktop has many inherent limitations. Typically, results files are simply written to a user's desktop and managing the results is left up to them. In some cases, running an analysis simply overwrites any previous analyses (making it very easy to lose data). A desktop environment makes it hard to collaborate with others because sharing results means sharing results files which are not easily shared when they are locked up in a desktop environment. Email and shared network drives are one way to address these issues, but they still involve a lot of manual work and discipline by users.

Another issue with the desktop is limited computing power. What if your analysis deals with uncertainty and requires a Monte-Carlo approach to simulating models. Or, what if you want to perform a large scale DOE or sensitivity study. There many types of analyses that could require hundreds, thousands or even more individual simulations. For even a high-end desktop environment, this could mean long wait times for analysts and this only aggravates the data management problem previously discussed.

In most organizations, a desktop application also means involvement with IT. The application has to be installed on the user's desktop and then updated when new releases come out. Getting the necessary IT resources to support installation across desktops is logistical aspect that must be dealt with and another drain on corporate resources.

## 4.2 Web-Based Client

To address many of these limitations, Xogeny has developed a set of tools for building web-based engineering analysis applications. These tools extract

information from FMUs and build a high-quality, intuitive user interface using this information.

As mentioned before, these FMUs frequently come from Modelica tools. As such, they include lots of useful information that can be automatically incorporated into the user interface. So, for example, such a web application can automatically incorporate logic to warn users about parameter values that are outside the expected limits. The nice thing about how these applications are built is that if the FMUs are updated, the information captured in them (parameters descriptions, default values, physical units, upper and lower limits) can be automatically incorporated into the application.

One might assume that a web browser is not an appropriate context for engineering applications because engineering applications require rich user interfaces with support for plotting, complex diagrams, detailed human machine interfaces or 3D animation. However, modern browsers have all this in the form of HTML5 support. Widely used web browsers like Firefox and Chrome can do all of these things seamlessly and without the need for any plugins.

A web-based application is easy to deploy and update because all that work can be done centrally once for all users. This reduces the impact on both the end user and the IT infrastructure to support installation and maintenance at the desktop. In fact, "installation" is typically as simple as circulating a URL to potential users. Furthermore, it is much easier to track usage and restrict access using a web-based approach.

In the case of the vehicle thermal management application, the web application can be found at the following URL<sup>2</sup>:

<http://vtm.demos.xogeny.com>

This user interface demonstrates that a high quality HTML5 user interface can be synthesized from information contained in an FMU. Such an interface can incorporate the necessary business logic to support non-expert users and operate in a way that is intuitive for most users.

But in order for this application to function, it must

<sup>2</sup>The "simulation" capabilities of the public web application have been removed. The results presented are actual simulate results, but they are simply injected rather than simulated. As a result, they are not affected by the parameter values in the UI.

be able to perform simulations. A web-browser generally has very limited access to the user’s desktop (certainly not enough to execute a simulation). So how does this interface perform the simulations required for the analysis? When the web application is compiled, support for simulation is compiled in through the `fmq.js` Javascript library. This library provides a native Javascript interface to the FMQ platforms cloud-based simulation capability. We’ll return to the simulation side shortly.

One last thing to note about the web-based user interface is the presence of a “History” view. This functionality is enabled by the **data management** features provided by FMQ. This view presents the user with a graphical history of their interactions and includes a tree structure of the different data sets that the user has either saved, simulated or is currently working with. It visualizes the relationships between each data set (which ones were derived from which). Hovering over a given version provides a summary of differences between that dataset and its parent dataset. All of this data is collected automatically and passively (*i.e.*, the user doesn’t have to do anything explicit or manually). Because this information is stored centrally, it would be easy to generate hyperlinks associated with each dataset visualized in the “History” view to share with colleagues. This kind of hyperlinking is the essence of the web but, unfortunately, it is not widely supported in the context of engineering tools.

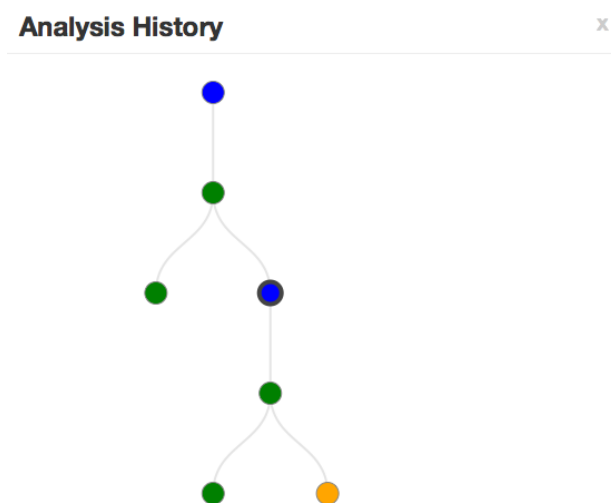


Figure 3: Visualization of data history

### 4.3 Cloud-Based Services

The central component of the FMQ platform (currently) is the ability to simulate an FMU. In this process, the FMU is registered in advance with the system and a RESTful web services API is used to request simulations involving specific parameter sets. This RESTful service can be accessed directly (to support large scale batch processing of FMU simulations) by making programmatic web requests or via a web-browser. For the web-based use case discussed in this paper, a special high-level Javascript library was developed that leverages the low-level RESTful API behind the scenes.

In addition to batch processing, the FMQ platform is capable of supporting interactive simulations. There are two sides to this interactivity. The first is the ability to stream simulation results interactively as the simulation is running. This means that clients can be asynchronously updated when new simulation results are generated. But it also means that the client can interactively feed **input** signals to the simulation while it is running as well. The result is the ability to create web-applications where the user manipulates the model and the model responds interactively.

Simulation results can be handled in several ways. We already discussed the ability to stream simulation results to the client as they are generated. In many cases, it may be preferable to simply wait until the simulation is complete and then provide the trajectory for key variables to the application. There are other cases where the web-based application may not have an immediate need for simulation results but it might wish to access them later. Finally, some types of analyses may generate large binary files as a by-product of the analysis (*e.g.*, a “meld” file from a simulation[8]). In those cases, the application may wish to access such artifacts. The FMQ platform supports all of these different use cases.

The FMQ services provide complete accountability for all data being processed. What this means is that for every analysis request being made, there is a record of the input data to the analysis and of all data generated from the analysis. Furthermore, the relationships between the job requests, job data, results and binary artifacts are all represented through FMQs Hypermedia API[9]. This provides a comprehensive approach to data management.

One final topic worth discussing is security. One of

the main concerns with any cloud-based solution is the security of the data. An important aspect of security is understanding exactly what “threat” is at issue and what techniques can be used to mitigate or eliminate those threats. A full discussion of such threats is beyond the scope of this paper but we are happy to discuss this topic. But it should be noted that the FMQ software is not tied to any particular platform or provider. As such, many of these security concerns can be addressed by **running the FMQ software on a computing cluster within the customer’s own internal network.**

## 5 Conclusion

This paper presents a representative engineering model and shows how that model can be shaped into an intuitive user interface with sufficient business logic and guidance to be used by a non-expert user. Such an application can directly leverage information already available from the FMU. The application can utilize an array of technologies already available in HTML5 to provide plotting, 3D rendering, sophisticated reports and human machine interfaces.

But more importantly, by leverage the capabilities already being actively developed and supported around web and cloud-based technologies, we can **improve** the engineering process providing better data management, greater opportunities for collaboration and data sharing as well as richer views of the engineering data and process by aggregating information that be automatically and passively collected through simple use of the application.

Ultimately, the goal of this work is to underscore the importance of model development. By providing an avenue for model developers to deploy their models to more users, we hope to create a positive feedback loop that will emphasize the value of model development and model-based system engineering that will, in turn, provide greater resources to model developers for creating high-quality, high-fidelity and high-impact engineering models.

This effort is really just the beginning. The FMI initiative helped to unify previously fragmented potential markets. The adoption of FMI has just started the process of building an eco-system around the FMI standard. FMQ is simply one example of how we can leverage the power of web and cloud based

resources and standardization to help drive improvements in the engineering world.

## References

- [1] Go Development Team. *Effective Go*. 2014. URL: [http://golang.org/doc/effective\\_go.html](http://golang.org/doc/effective_go.html).
- [2] Caleb Doxsey. *An Introduction to Programming in Go*. Caleb Doxsey, 2012. URL: <http://www.golang-book.com/10#section2>.
- [3] Rich Hickey. *Clojure core.async Channels*. 2013. URL: <http://clojure.com/blog/2013/06/28/clojure-core-async-channels.html>.
- [4] Jamie Allen. *EffectiveAkka*. O’Reilly Media, 2013.
- [5] PromisesA+ Organization. *PromisesA+ Specification*. 2013. URL: <http://promisesaplus.com/>.
- [6] John Batteh, Jesse Gohl, and Sureshkumar Chandrasekar. “Integrated Vehicle Thermal Management in Modelica: Overview and Applications”. In: *Proceedings of the 10th International Modelica Conference* (2014).
- [7] MODELISAR Consortium. *Functional Mockup Interface, Version 1.0*. 2010. URL: [https://svn.modelica.org/fmi/branches/public/specifications/FMI\\_for\\_ModelExchange\\_v1.0.pdf](https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_v1.0.pdf).
- [8] Michael M. Tiller and Peter Harman. “recon – Web and network friendly simulation data formats”. In: *Proceedings of the 10th International Modelica Conference* (2014).
- [9] Roy T. Fielding. *REST APIs must be hypertext-driven*. 2008. URL: <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.





# recon – Web and network friendly simulation data formats

Michael Tiller  
Xogeny Inc., USA  
michael.tiller@xogeny.com

Peter Harman  
CyDesign Ltd., UK  
peter@cydesign.com

## Abstract

There are many different commonly used file formats for storing time series data. Most of these file formats are designed with the assumption that the file itself will be locally available to the software that will be reading or writing the data stored in them. While this assumption is an excellent one for desktop based tools with direct access to disk drives capable of moving virtually instantaneously around from sector to sector, there are a growing number of applications for which local access is not necessarily available. For these applications, we've initiated the recon project to develop more suitable formats.

With the emergence of web and cloud based modeling and simulation technologies, the time has come to explore file formats specifically optimized for non-desktop applications. In this paper, we present a new set of file formats that are specifically designed for web and cloud based approaches. This paper reviews the key requirements for web and cloud enabled applications and then presents a specification for two file formats that address those requirements.

When considering the various use cases that drove our requirements, we recognized that two different file formats were actually required. The first format, the `wall` format, is optimized for writing. The other format, the `mold` format, is optimized for reading over a network (*i.e.*, minimizing the number of reads and bytes read). We will discuss the layout of each of these formats and describe the use cases for which they are most appropriate.

In the open tradition of the Modelica Association, the authors have made specifications and implementations for these formats available as open source libraries with the hope that they will benefit the community as a whole.

**Keywords:** *Modelica, FMI, simulation results, cloud, web, open source*

## 1 Introduction

Several groups have examined the issue of standardized file formats[1, 2] in the context of Modelica. In keeping with the principles of the Modelica Association, an ideal choice would be a production ready format that is open source and cross-platform. With these requirements in mind, most people consider HDF5 a natural choice. There are already open source implementations and the file format has been widely used. In fact, it has even been adopted by The MathWorks for use in MATLAB.

But HDF5 has some practical drawbacks. The first is that it is not truly cross-platform. The reference implementation of HDF5 is written in C. The implementation is primarily targeted for use within C, C++ or Fortran applications. While there are various libraries available for reading HDF5 on the Java platform[3], they are incomplete and awkward to use.

Another issue with HDF5 is that for simple time-series data it is over engineered. HDF5 is feature rich, of that there is no question, but these features come at the cost of complexity. This is why you see very few implementations outside of the reference implementation. Furthermore, the file format makes extensive use of “seek” operations and assumes they are relatively inexpensive. This assumption is reasonable if you are able to communicate directly with the hard drive that the files are stored on, but it isn't reasonable when these files are only available through the network.

There are, of course, many standards for encoding data in web or cloud based environments. The most popular formats, by far, are Javascript Object Notation (or JSON, for short) and XML. There are other approaches as well like Google Protocol Buffers[4], Avro[5] and Thrift[6]. Different approaches have different goals. Some define schemas to add a level static checking. Others exist mainly to compress the data being transmitted. Finally, others exist to introduce a layer of interoperability with RPC frameworks or “big data” tools like Hadoop[7] and

Storm[8].

So where does that leave us? Should we adopt the tried and true standards from the engineering world and simply live with their lack of interoperability with important platforms like Java or Javascript and poor performance when accessed remotely? Or, should we adapt tools from the “big data” world, that were developed for quite different use cases, to work in the engineering world.

In some sense we’ve chosen a compromise. As we will see shortly, the `wall` and `meld` formats are fundamentally derived from the `msgpack`[9] specification. This gives us excellent cross platform compatibility. But `msgpack` is simply a serialization protocol. To address some of our more important concerns, to be discussed shortly, we needed to design a format the imposed additional structure on top of `msgpack`. So in this sense, we’ve created a set of original file formats that leverage open standards, like `msgpack`, but re-purpose them for modeling and simulation applications.

## 2 Goals

After independently reviewing various file formats, the authors were not happy with the existing options for web and cloud based modeling and simulation. The `recon` project started as a discussion about requirements. For our applications, the following requirements were identified:

### 2.1 Requirement 1 - Adding Data

Simulations are constantly producing additional data. For this reason, adding new data to an existing file is an operation performed many times during a simulation. For this reason, adding data to an existing file should be fast and easy. The key thing is to avoid having to rewrite previous data or, even worse, move data around within the file. For this reason, the ideal solution is to have the ability to simply append new data at the end of the file.

### 2.2 Requirement 2 - Minimizing I/O

In web and cloud based applications, it is not always practical to download the complete set of results for a simulation into the browser environment. There are many use cases where it would be best to be able to access results “on demand”. In these environments, such requests for data will be done via HTTP[10].

However, each of these requests will come with far greater latency than a simple request to read from a disk drive and far less bandwidth. As such, we would want to minimize the number of such requests and the amount of data necessary to transmit in each request. This means we need a way to “cluster” the data we are interested in so as to minimize both the number of requests required and the amount of data in each response.

### 2.3 Requirement 3 - Cross Platform Support

The file formats developed as part of the `recon` project are targeted at web and cloud based applications. Client side web programming is dominated by Javascript. On the other hand, server side programming is done in a wide variety of languages (*e.g.*, Java, Python, Javascript). Meanwhile, numerical analyses such as simulation are typically done in languages like C, C++ and FORTRAN. For this reason, it should be possible to implement libraries in all of these languages for generating and extracting simulation results.

### 2.4 Requirement 4 - Aliasing

One of the common patterns in component-oriented modeling approaches like Modelica is that many variables end up with exactly the same solution trajectories. When storing simulation results from such tools, it is useful to recognize that considerable disk space can be saved by recognizing the fact that these variables all share a common underlying solution trajectory. Typically, the data for each unique solution trajectory is stored once and each variable is simply a reference to the underlying solution trajectory. Even more storage can be saved by recognizing that some trajectories are related to other trajectories by very simple transformations (*e.g.*, a simple sign change). For this reason, it is very useful if these kinds of relationships are directly represented in the file format itself.

### 2.5 Requirement 5 - Data Types

When dealing with simulation results that come from the solution of differential equations, the main type of result is a solution trajectory. In these cases, both the dependent and independent variables are typically represented as floating point numbers.

But these are not the only types of data a simulation or other numerical analysis might yield. From the Modelica world, we might easily have results that are either reals, integers, booleans or strings (since these are all fundamental built-in types in Modelica). But why not hierarchical data structures (as represented by records in Modelica) as well?

## 2.6 Requirement 6 - Metadata

One issue with data files is that if you don't provide a means for associating metadata with entities in the file, the metadata will **become** entities in the file. For this reason, we deemed it important that metadata should be treated in a "first-class" way. Specifically, it should be possible to associate metadata with the file as a whole and with data structures in the file all the way down to individual signals. This would allow tools to persist other important information, beyond the solution, in these data files. For example, information about common plots or plotting options, descriptions of the signals, units or display units could all be managed in a structured way without being confused with data and without needing to be a formal part of the file format specification.

## 2.7 Requirement 7 - Hierarchy

Many tools create structures that are hierarchical. In the Modelica world, we have deep hierarchies of instances in simulated models. We also have hierarchies for packages and the definitions contained in them. So it is important that a file format can represent these hierarchies in some way.

In our experience, trying to organize results according to an instance hierarchy creates quite a bit of complexity. While tools could exploit some of the previous requirements (primarily 5 and 6) to achieve a hierarchical representation, we've found that simply encoding hierarchy in the names of variables (e.g., `car.engine.crankshaft.tau`) is typically sufficient and can avoid considerable complexity.

## 2.8 Requirement 8 - Easy Translation

Even though our goal is to have a format that is well suited to web and cloud based applications, it should also be capable of representing the kinds of simulation results we interact with in a desktop environment. For this reason, one of our requirements was the ability to translate data in the "dsres" format into

the recon formats. Such a translation should preserve all data and metadata normally associated with the "dsres" format. Furthermore, the resulting recon files should be approximately the same size as the original "dsres" file.

# 3 Approach

## 3.1 Reading vs. Writing

In reviewing these requirements, the main design challenge was trying to reconcile requirements 1 and 2. Implementing requirement 1 typically involves the need to write data out one row at a time, where each row represents the values of all the variables for successive solution times. As such, the solution values for any particular *variable* are widely spaced. However, requirement 2 requires us to be able to extract a given variable with a minimum number of I/O operations. In other words, requirement 1 typically results in data being fragmented while requirement 2 depends on that data being clustered together.

Our solution to this design problem was to design two file formats. The first, the `wa11` format, is designed for writing. Not only does it make adding data fast and efficient, it also supports, unlike the `dsres` approach, adding data for multiple tables at once. In practice, this means that if you have variables in your simulation that are partitioned such that they have different independent variables (as with the new clock semantics in Modelica 3.3), this format supports writing out new data for any of these variables. In other words, you can add results that may have completely different time bases. You can also include results from multiple simulations and use the metadata features to associate the specific parameter sets with each table.

The other format, the `me1d` format, is optimized for reading. Specifically, it is optimized for requirement 2. For an ideal format, it should be possible to extract a single result trajectory in a single read. This would act to minimize the impact of latency. Furthermore, the bytes read should contain only data associated with the desired signal. This would minimize the impact of limited bandwidth. As we will discuss shortly, the `me1d` file manages to achieve these performance characteristics for all but the first signal read.

Our expectation is that tools will write data out (during simulation) in the `wa11` format. Tools may choose to keep the data in this format. For platforms

where network access is not a requirement, the write optimized nature of the `wall` format will probably be adequate for both reading and writing. However for cases where data will be read over a network, we expect that tools will, upon completing a simulation, rewrite their data into the `me1d` format.

### 3.2 Serialization

There are really two aspects to each `recon` format. The first is the structure of the file (where different pieces of information reside in the file, something we'll discuss in Section 4) and the other is how the actual data is represented.

Obviously, the data is represented as individual bytes. So we must define the process by which multi-byte pieces of information (*e.g.*, floating point numbers) are “serialized” into bytes. One of the implicit goals of this project was to make a file format that was easy to read and write. Since serializing and deserializing data was a big part of the implementation, we could make the implementation much easier if we leveraged existing standards for serialization and deserialization.

One of the interesting things about coming from the web and cloud based application side is the ubiquity of JSON notation. While the Javascript language itself has many “unusual” semantics, the syntax and semantics around serialization and deserialization are surprisingly simple and intuitive. Unlike XML, for example, writing a parser for JSON and then mapping into a native language representation is surprisingly easy and widely supported.

However, JSON is a textual representation. The problem with a textual representation is the additional overhead of having to parse and interpret the text and convert, without any loss of precision, into a binary representation. For this reason, we didn't consider JSON by itself a practical approach to serialization and deserialization.

While we wanted to avoid the parsing aspect of JSON, the JSON data model[11] is well suited to our purposes and many different groups have attempted to create a binary representation that follows the JSON data model. So our initial approach was to consider BSON [12] which is a binary format that is formally specified and widely implemented because it is one of the cornerstones of the MongoDB database[13].

Unfortunately, the BSON serialization scheme has a significant drawback. The way it serializes arrays is very space inefficient. This is because JSON itself

supports sparse indexing of arrays. As a result, a serialization must include, for each value in the array, the index as well. This adds significant overhead. There is no way to specify that all elements in the array are sequential. As such, there is no way to avoid this significant penalty.

Fortunately, our reference implementation in Python[14] had a clean separation between the serialization scheme and the structural aspects. This made it very easy for us to experiment with other serialization techniques. We investigated other similar serialization schemes like Smile[15], BJSON[16] and UBJSON[17]. However, none of them seemed to have a critical mass behind them. Indeed, there doesn't seem to be a consensus in the JSON community on how to serialize JSON in a binary form.

As part of our investigation, we also looked at `msgpack`. It turned out that, like BSON, `msgpack` was formally specified and implemented for a wide variety of platforms[9]. Furthermore, in testing `msgpack`, we found that it had much better storage efficiency compared to BSON. So, in the end, we moved forward using the `msgpack` serialization scheme.

The `msgpack` approach had a couple of unanticipated benefits. In `msgpack`, floating point numbers can be encoded in either single or double precision representations. Also, the specification identifies and formally specifies several different optimizations to minimize the number of bytes required to store short integers or short strings. The underlying “types” permitted in this format map very easily into the JSON format which, in turn, means that it maps well into the native data types common across all the languages we are interested in. Finally, the `msgpack` serialization scheme includes an extension mechanism for including additional data types beyond those in the specification. While we don't have any immediate use for these extensions, it is nice to have that feature if we ever find that `msgpack`'s serialization is too constraining.

## 4 Specification

With the motivation behind us, let's turn to the actual specification of these formats. In this section we will describe the layout of both the `wall` format and the `me1d` format. As mentioned in the previous section, the serialization is done using `msgpack`. So we will focus mainly on the layout of data within the file. When describing the actual data being stored we will

use JSON notation to document the data with the implicit understanding that this data will be serialized and deserialized using `msgpack`.

Before we get into the specifications for each of these formats, it will be useful to discuss a few topics that are relevant to both formats. For example, what exactly are we storing in these file formats? Both formats support the storing of tables and objects. Tables are, as the name implies, a structure for storing tabular data. It is worth noting that there are no restrictions on what kind of data can be stored in a table except those imposed by the underlying `msgpack` format (which are very few). This means tables can mix integers, floats, doubles, longs, short ints, strings, booleans and even objects across different columns in the same table.

In addition to tables, we can register objects to be stored in our results file. These are essentially free-format pieces of data. The objects can be used to create arbitrarily deeply nested data structures that mix all varieties of data. Once again, the only limitations are those imposed by `msgpack` and/or the source language.

When storing tables and objects in a file, they must be referred to by names. The same namespace is used for both objects and tables (in other words, you cannot have a table with the same name as an object). It also means that no two tables and no two objects can share the same name either. The columns of each table are also named and no two columns within the same table can share the same name. But there is no such restriction between columns in different tables (or fields in different objects, for that matter).

Finally, it is worth noting that certain pieces of data are optional. In those cases, we have consistently followed a policy of leaving out both the key and the value. In other words, it is not sufficient to simply associate a null value with a key. **Both the key and the value must be removed** if there is no value provided. The guiding principle here is that parsers should not have to do excessive amounts of null value checking.

With that background out of the way, let's proceed with our explanation of the two file formats.

## 4.1 Wall Format

Recall that the `wall` format is optimized for writing and that this, in turn, means being able to easily add data. You can think of the `wall` format as being similar to a brick wall where each brick (new piece of data) is staggered with respect to others. As you will

see, we are **not** storing information in homogenous arrays and this means that we cannot predict the index of data simply based on information about which row or column it is in. Also note that it is possible to have data from two different tables interleaved between each other. This allows us to add data with two distinct time bases. But it makes the location of data even more difficult to predict. However, remember that the `wall` format is optimized for writing, not reading and that if network access is required then tools will typically rewrite their data into the `meld` format.

### 4.1.1 Leading Bytes

Each `wall` file starts with the following sequence of bytes:

```
0x72 0x65 0x63 0x6f 0x6e 0x3a 0x77
0x61 0x6c 0x6c 0x3a 0x76 0x30 0x31
```

This is a hex encoding of the ASCII string `recon:wall:v01`. This allows us to identify whether this is a `recon wall` file and, if so, what version of the specification should be applied.

The next four bytes are a binary encoding of the length of the header. This encoding is done in so-called "network byte order" (big-endian). The byte encoding of the length is not considered part of the header (*i.e.*, the length indicated doesn't include the 4 bytes that encode the length).

### 4.1.2 Header

Once the length of the header is known, the bytes for the header are read in. These bytes are assumed to have been serialized in `msgpack` format so we must next unpack (deserialize) these bytes. Once unpacked, the header should contain the following information:

```
{
  "fmeta": {<file-level metadata>},
  "tabs": {
    "<table name>": {<table data>}
  },
  "objs": {
    "<object name>": {<object metadata>}
  }
}
```

The "fmeta" key is associated with the value for any file level metadata. This metadata is, itself, represented in our notation here as an object in JSON but will be encoded as a map in `msgpack`. The "tabs"

key is associated with a value that maps the names of tables to table data. The format of this table data is as follows:

```
{
  "tmeta": {<table-level metadata>},
  "sigs": [<list of signals>],
  "als": {
    "<aliasname>": {
      "s": <base signal name>,
      "t": <transform string> // OPTIONAL
    }
  },
  "vmeta": {
    "<varname>": {
      <variable-level metadata>
    } // OPTIONAL
  }
}
```

The "tmeta" key is associated with metadata (again, represented as a msgpack map) but this time it is metadata associated with the table. In addition, we have the "sigs" key which represents an ordered list of signals. A signal represents an actual solution trajectory and the order is important because the order in this list indicates the order in which the data will be stored in successive rows (to be discussed shortly).

The "als" key represents any aliases present in the table. Again, this is a msgpack map where the name of the alias is the key. There are two essential pieces of information associated with each of the aliases. The first, stored under the "s" key (which stands for "signal") is the name of the base signal that this alias is based on. The "t" key is used to represent the transformation that should be applied to the base signal to compute the value of the alias signal. Note that this transformation is **optional**. The possible values will be described shortly in 4.3.

Finally we have the "vmeta" key, which is a map where the keys are the names of variables (*i.e.*, both signals and aliases) and the values are any metadata (again, stored as a msgpack map) associated with the named variable. Note that keys are only present in the "vmeta" map if there is metadata associated with that variable.

Returning to the header level entries, the "objs" key is associated with a value which is, in turn, a msgpack map. Each key in that map represents an object name and the value associated with those object name keys represents the metadata associated with the named object.

### 4.1.3 Entries

Following the header, the remainder of the file consists of "entries". Each entry is preceded by 4 bytes in network byte order indicating the length of the entry (again, the length indicated does not include the 4 bytes used to represent the length). All entries are encoded maps in msgpack format. There are two types of possible entries and there are no rules about which can be present (*i.e.*, they can appear in any order and be interleaved).

The first type is a "row entry" which details a new row for a specified table. The format of a row entry is as follows:

```
{
  "<table name>": [<list of signal values>]
}
```

where the table name must be a key in the "tabs" map found in the header and the order of values in the list of signal values must correspond to the order defined by the list associated with the "sigs" key value within that table.

The other entry type is a "field entry". These represent updates to the values of fields in objects and have the following format:

```
{
  "<object name>": {
    "<field name>": <field value>,
    "<field name>": <field value>
  }
}
```

Note that the header does not specify which fields are present in the object. This can only be determined by processing all the field entries and incorporating fields as they are given values. The complete value for a given object is determined by processing all field entries associated with that object in the order they appear in the wall file. The complete value is then simply the final value after all processing is completed. One consequence of this is that it is therefore possible to have the values of individual fields change while writing the file.

### 4.1.4 Wall Format Summary

The following outline attempts to summarize all the details presented so far:

```
// ID
0x72 0x65 0x63 0x6f 0x6e 0x3a 0x77
0x61 0x6c 0x6c 0x3a 0x76 0x30 0x31
```

```
// Header length, network order
0x?? 0x?? 0x?? 0x??
{
  "fmeta": {<file-level metadata>},
  "tabs": {
    <table name>: {
      "tmeta": {<table-level metadata>},
      "sigs": [<list of signals>],
      "als": {
        <aliasname>: {
          "s": <base signal name>,
          "t": <transform string> // OPTIONAL
        }
      },
    },
    "vmeta": {
      <varname>: {
        <variable-level metadata>
      } // OPTIONAL
    }
  },
  "objs": {
    <objname>: {<object metadata>}
  }
}

// Followed by zero or more entries
// which can be either...

// ...field entries...
0x?? 0x?? 0x?? 0x?? // entry length
{
  <object name>: {
    <field name>: <field value>,
    <field name>: <field value>
  },
}

// ...or row entries
0x?? 0x?? 0x?? 0x?? // entry length
{
  <table name>: [<list of signal values>]
}

```

where all maps are encoded in msgpack.

## 4.2 Meld Format

### 4.2.1 Leading Bytes

Each meld file starts with the following sequence of bytes:

```
0x72 0x65 0x63 0x6f 0x6e 0x3a 0x6d
0x65 0x6c 0x64 0x3a 0x76 0x30 0x31
```

This is a hex encoding of the ASCII string `recon:meld:v01`. This allows us to identify whether this is a recon meld file and, if so, what version of the specification should be applied.

The next four bytes are a binary encoding of the length of the header. This encoding is done in so-called “network byte order” (big-endian).

### 4.2.2 Header

Once the length of the header is known, the bytes for the header are read in. These bytes are assumed to have been serialized in msgpack format so we must next unpack these bytes. Once unpacked, the header should contain the following information:

```
{
  "fmeta": {<file-level metadata>},
  "tabs": {
    "<table name>": <table data>
  },
  "objs": {
    "<object name>": <object data>
  },
  "comp": true|false // Compression flag
}
```

This is very similar to the wall format presented in Section 4.1. Again, we see file level metadata exactly as it is used in the wall format. We also have the “tabs” and “objs” keys, also present in the wall format but with an important distinction which is that the values that follows them have a different format, as we shall see shortly.

But we also have a new key, the “comp” key, which isn’t present at all in the wall format. The value associated with the “comp” key indicates whether the remainder of the file (after the header) is not just encoded (using msgpack) but also compressed. If the value associated with the “comp” key is true, then all remaining msgpack encodings present in the file after the header are compressed using bz2[18] compression.

For reasons that will become obvious, the data for tables and objects is different in the meld header than in the wall header. In a meld file, the table data has the following format:

```
{
  "tmeta": {<table level metadata>},
  "vars": <list of variable names>,
  "toff": {
    "<varname>": {
      "i": <index of variable data>,
      "l": <length of variable data>,
      "t": <transform string> // OPTIONAL
    }
  },
  "vmeta": {
    "<varname>": {
      <variable level metadata>
    }
  }
}
```

```

    } // OPTIONAL
  }
}

```

The "tmeta" is, again, the table level metadata. Similarly, the "vmeta" key is associated with variable level metadata which is a map where the variable name is the key (again, only present if there is metadata associated with the specified variable) and the associated value is the variable level metadata.

The "vars" key is associated with an ordered list of the variables present in the file. The "toff" key is associated with a map that specifies important information about the location of the variables within the file. It is the "toff" data that makes it easy for us to extract individual signals. The "i" key is associated with the starting byte, within the file (starting from 0), of the data associated with the variable and the "l" key is associated with the length of that data. The optional "t" key defines the transformation, if any, to be applied to the variable data (see Section 4.3 for more details).

Returning to the header data, the object data associated with the "objs" key has the following format in a meld file:

```

{
  "ometa": {<object level metadata>},
  "i": <index of object data>,
  "l": <length of object data>
}

```

The "ometa" key is associated with the metadata of the associated object. The "i" and "l" keys are used just as they are within tables, to define the index and length, respectively, of the object data within the file.

#### 4.2.3 Variable Data

As discussed in Section 4.2.2, both variables (conceptually, columns in tables) and objects have an offset and a length provided in the header. In the case of a variable, the data that is extracted from that location in the file will be a **list** of values in msgpack format. The values in that list represent the values for the specified solution variables (first row first, last row last). In the case of an object, the data that is extracted from that location in the file will be a **map** where the keys in the map represent the fields present in the object and the values associated with those keys are the field values.

#### 4.2.4 Header Size

It is worth pointing out that when writing the file, the exact length of the header (when encoded in msgpack format) cannot be known *a priori* (we shall explain why, shortly). For this reason, a collection of bytes representing the largest possible size must be reserved for the header. The size of the header depends on the number of tables and objects as well as the number of signals in each table so it important that all of this information is known before determining the maximum number of bytes required to represent the header.

However, when the final version of the header is written out (once the location of all the data can be determined), its size may be less than originally anticipated. This is a result of the msgpack format's aggressive compression of small integers. For this reason, there may be a few unused bytes present between the end of the header and the first variable or object data in the file. While it is true that a few bytes will be wasted as a result, it really only means that msgpack's aggressive optimizations will be wasted in this case (and this case only).

#### 4.2.5 Meld Format Summary

The following outline attempts to summarize all the details presented so far:

```

// ID
0x72 0x65 0x63 0x6f 0x6e 0x3a 0x6d
0x65 0x6c 0x64 0x3a 0x76 0x30 0x31
// Header length, network order
0x?? 0x?? 0x?? 0x??
{
  "fmeta": {<file-level metadata>},
  "tabs": {
    "<table name>": {
      "tmeta": {<table level metadata>},
      "vars": <list of variable names>,
      "toff": {
        "<varname>": {
          "i": <index of variable data>,
          "l": <length of variable data>,
          "t": <transform string> // OPTIONAL
        }
      },
      "vmeta": {
        "<varname>": {
          <variable level metadata>
        } // OPTIONAL
      }
    }
  },
  "objs": {
    "<object name>": {
      "ometa": {<object level metadata>},
      "i": <index of object data>,

```



```

    "l": <length of object data>
  }
},
"comp": true|false // Compression flag
}

// Followed by any padding
// resulting from header shrinkage

// Followed by zero or more blocks
// of msgpacked data representing
// either vectors or objects whose
// offsets and lengths are specified
// in the header)

```

### 4.3 Transformations

In the previous sections, there were several mentions of a so-called “transform string”. This is an optional piece of information associated with an alias (in the case of the `wall` format) or a variable (in the case of the `meld` format). It defines the transformation, if any, that must be performed over some base data in order to retrieve the true value of the referenced data. There are presently only two allowed transformation types that are supported by the `recon` formats.

The first transformation type is the “inverse” transformation. This transformation is indicated when the transform string has a value of `"inv"`. The impact of the inverse transformation depends on the data type. For numeric data types, the inverse transformation causes the data to have its sign inverted. For boolean data, the inverse transformation applies a logical not operation to the data. With this simple transform alone, it is possible to avoid storing a significant amount of data.

The other transform type is the “affine” transform. This transformation is indicated when the transform string has a value of `"aff(s,o)"`, where `s` represents a scale factor and `o` represents an offset value. In the presence of this transformation, all **numeric** values in the base data should be multiplied by the scale factor, `s`, and then added to the offset value, `o`. As one reviewer of this paper noted, unit transforms are almost always affine in nature. This means that the affine transformation permits results to be stored in many different physical units without taking up any appreciable additional storage.

No transform should be applied to the data if:

- No transform string is present
- The transform string is unrecognized/cannot be parsed

- The transform does not apply to the underlying data type (e.g., applying the affine transformation to a Boolean value)
- There was an kind of error or exception when attempting to apply the transformation

In all but the first case, the tool or environment is strongly encourage to provide an error message alerting the user. Tools are also free to treat all but the first of these conditions as an error and suppress access to the alias data.

## 5 Discussion

### 5.1 Use Case

Although it is implicit in the requirements listed in Section 2, it is worth elaborating a bit more on the specific use case that drove these requirements.

For web and cloud based simulation, the bulk of the computational work is done remotely. In cloud services, there is an effect sometimes called “data gravity” which dictates that to improve overall throughput, the computing platforms tends to gravitate to where the data is stored (*i.e.*, the same service provider or at least ones that are connected with low latency, high bandwidth connections).

So if one seeks an optimal configuration where the computing and storage are well connected, it is easy to store the complete simulation results without any significant concerns for latency or bandwidth.

The complication comes from having to access that data via a “normal” network connection. A typical use case (and the one that we imagined while formulating our requirements) is one where a web application, running in a web browser, needs to display simulation results. The question then is how could such an application make effective use of simulations results generated “in the cloud”?

One approach to this could be for the web browser to download the complete simulation results. Without consideration for web and cloud based applications, such results might very likely be stored in formats like HDF5 or MATLAB version 4 format. The first problem is that these formats are not well supported in a Javascript environment. In fact, the authors are not aware of a single Javascript implementation that can read either format.

Even if you had Javascript implementations for these formats, this approach would necessitate having to download the entire file into some kind of “in

memory” representation to be parsed. This is because these formats do not provide a simple way to extract the required data via a few simple reads.

The other approach, the one we’ve taken for the recon project, is to use a format (of our own design, the me1d format) that is designed for remote access. Given access to header information (more on this in a moment), we can extract a single table column or single object with a single read. Furthermore, we can use the HTTP Range header to specify exactly which range of bytes we require. In this way, we only have to endure the latency of a single network request and we avoid transmitting any extraneous data which minimizes the impact of bandwidth limitations.

Note the previous paragraph presumes we already have the header information from the files. The worst case scenario for getting header information is to make two additional (one time only) network requests. The first request would be for the first 18 bytes (again, utilizing the Range header) to establish the size of the header. The next request would be to read the binary header data which includes information about the locations of all table columns and objects. Once these two requests have been made, the client would be able to access any object or column with only one additional request.

One of the things we’ve tried to do in this project is avoid solving problems that have already been solved. This was the motivation behind the use of msgpack, but we are also assuming that most network requests for data will be made using HTTP. This is a safe assumption given the ubiquitous nature of HTTP (or HTTPS, for that matter). But if we assume that requests are made via HTTP we also gain the additional benefits of caching.

Most networks are already equipped to efficiently handle caching transparently. Of course, within a given application, we might maintain a cache of headers for different results files that we may be interested in. But what about multiple users accessing these results across the same network? As it turns out, once one user accesses the file, it is quite likely that the header information will be stored in a caching proxy between the users network and the storage provider. This is a caching scenario that we cannot address within an application, but nevertheless, it is very likely that in such a multi-user environment the existing network proxies would transparently act to improve overall performance and enhance the user experience.

## 5.2 Metadata

As discussed throughout Section 4, the wall and meld formats have extensive support for metadata. This provides a clean mechanisms for including metadata in files without the need to mix it in or conflate it with actual data. Furthermore, metadata is supported for a wide range of entities represented in the file.

There are many potential applications for such metadata. For example, file level metadata can be used to store useful information such as who ran a simulation, what particular model they ran, what modifications (if any) were applied to the model or even a complete listing of all the parameter values that were used to simulate the model. In addition, general experimental data could also be stored in the results file.

Metadata at the signal level could be used to specify not just descriptions of those variables along with their physical units but, could also be used to include other attributes like start values, nominal values, *etc.*,

In summary, the recon formats make metadata a first class citizen within the file formats and this opens the door to many very useful applications involving metadata.

## 5.3 Performance

It is worth taking a moment to discuss the actual performance of this approach vs. existing approaches. As a baseline, we have generated a “representative results file” by simulating the R3 robot example from the Modelica Standard Library. This results file, produced by Dymola and written in the “dsres” format, will serve as our baseline.

The first issue we will examine is space efficiency. The baseline results take up 3,069,623 bytes of storage (for the storage options we selected). When stored in the me1d format **without compression** those same results take up 3,169,994 bytes. Note that this translation process from the dsres format to the me1d format preserves variable names and description strings as well as numerical trajectories. This means the me1d format is only 3.3% larger than the corresponding dsres file. If we enable compression, the me1d file is only 2,787,467 bytes which means it is almost 10% smaller than the dsres file.

So, in terms of storage, the me1d format is quite comparable to the dsres format. At first, this may seem counter-intuitive since we know, from our earlier discussion of msgpack, that arrays of floating

point numbers incur an extra byte for storage. However, the dsres format cannot store strings very efficiently and that long descriptions will result in enormous amounts of padding. So it would appear that, on net, these two effects cancel each other out. However, one advantage that the me1d format has that the dsres format doesn't (and an advantage that is not capitalized on in these benchmarks at all) is the wider range of alias transformations. In particular, it is possible to relate two signals via an affine transformation with the me1d format. Unfortunately, we do yet have any data on the potential savings this might offer.

Another benchmark worth considering is the time it takes to extract one particular trajectory from a results file. For extracting results from a dsres file, we used the `scipy.io.loadmat` function to load the `data_2` matrix and then extracted the 0<sup>th</sup> column (Time). In other words,

```
mat = loadmat("tests/fullRobot.mat")
T2 = mat["data_2"]
time = T2[0,:]
```

Running this script 5 times gave the following times (in milliseconds): 18.111, 18.143, 20.076, 19.309, 21.773 for an average time of 19.482 milliseconds.

In the case of the me1d file, we opened the results file, created a `MeldReader` object to read it, extracted the `data_2` table (called T2 in the translated file) and then extracted the data associated with the Time signal. That code looks as follows,

```
with open("dsres_robot.m1d", "rb") as fp:
    meld = MeldReader(fp)
    dt = meld.read_table("T2")
    time = dt.data("Time")
```

Running this script 5 times gave the following times (again, in milliseconds): 17.260, 18.074, 16.882, 14.608, 14.572 for an average time of 16.280 milliseconds.

It is worth noting the SciPy implementation is very mature and utilizes numpy internally. So one would expect excellent performance from that implementation. Nevertheless, extracting data based in the me1d format was over 15% faster on average.

The last benchmark will be reading multiple signals. In fact, our benchmark in this case will be to extract all transient signals into a Python dictionary. We will do this again for both formats. We used the following script to extract the same signals from the dsres file format:

```
ret = {}
mf = dymat.DyMatFile("tests/fullRobot.mat")
for signal in mf.names(2):
    ret[signal] = mf.data(signal)
ret["Time"] = mf.abscissa(2)
```

This code uses the `dymat` library which, in turn, leverages SciPy and numpy. Running this code 5 times gave the following execution times (in milliseconds): 519.50, 495.38, 483.60, 476.69 and 481.15 for an average execution time of 491.26 milliseconds.

The script for performing this benchmark using the me1d format looks as follows:

```
ret = {}
with open("dsres_robot.m1d", "rb") as fp:
    meld = MeldReader(fp)
    dt = meld.read_table("T2")
    for signal in dt.signals():
        ret[signal] = dt.data(signal)
```

Running this script 5 times, we record execution times (in milliseconds) of: 246.34, 240.66, 225.58, 224.22, 222.65 for an average of 231.89 milliseconds (over 50% faster compared to the dsres version of the benchmark).

In fairness, it is worth pointing out that we generally expect simulation tools to write output results in the wall format first and then convert them into the me1d format. As such, they will incur some penalty as a one-time cost when regenerating their results in the me1d format and this penalty is not taken into account here. Furthermore, this benchmark doesn't include any writing performance benchmarks (primarily because we have not connected any of these codes to actual simulation tools to measure write performance).

In summary, we do not have any benchmarks that compare write performance. But in terms of storage efficiency, the me1d file format was only 3% larger when compared to our baseline results in dsres format. On the other hand, in terms of read performance, the me1d format was between 15 and 50% faster depending on the amount of data to be read.

## 5.4 Implementations

As already mentioned, there is a reference implementation of both the wall and me1d formats already available in Python[14]. There are also implementations available for both C[19] and Java[20].

In addition, the wall format is also now supported by OpenModelica as of r18784. The implementation of the wall format required 408 lines

of code to implement in OpenModelica (this includes implementation of all necessary msgpack operations, *i.e.*, no external libraries are used to implement msgpack functionality) while the implementation of the “dsres” format uses 656 lines of code.

## 6 Conclusion

In conclusion, the recon file formats support many important features:

1. **easy to implement on a wide range of platforms** - An open source reference implementation is available in Python with implementations in C and Java already planned.
2. **efficient disk and network access** - These formats have been optimized for efficient network performance. But as the benchmarks show, these formats also perform very well for when results are stored locally on hard disks.
3. **first-class metadata** - With metadata available at all structural levels, results files can embed metadata in a clean and practical way. This has the potential to open up many new and interesting applications.
4. **tables containing mixed data types** - All data types are given equal treatment within the recon formats. As a result, it is possible to embed a wide variety of data and still maintain all the other benefits associated with these formats.
5. **richer alias transformations** - The recon formats support not only the common “inverse” transform, but a richer set of affine transformations. This could lead to further space efficiencies.
6. **efficient treatment of strings** - MATLAB version 4 files are very inefficient for storing collections of strings. By using msgpack for serialization, we get very efficient handling of string data.
7. **ability to store objects/structures** - Not all data is tabular. For example, parameter data used as input to simulations is more naturally represented as an object. There are many other examples of potential applications that need to represent data as objects. The recon formats even allow these objects to be embedded within tables.

The recon formats provide all these benefits without any significant compromise in storage efficiency.

## References

- [1] A. Pfeiffer, I. Bausch-Gall, and M. Otter. “Proposal for a Standard Time Series File Format in HDF5”. In: *Proceedings of the 9th International Modelica Conference (2012)*. URL: [http://www.bausch-gall.de/ecp12076495\\_PfeifferBausch-GallOtter.pdf](http://www.bausch-gall.de/ecp12076495_PfeifferBausch-GallOtter.pdf).
- [2] Jörg Rädler. “DyMat - HDF5 export and other Modelica/Python projects”. In: (2013). URL: <http://www.j-raedler.de/2013/01/dymat-hdf5-export-and-comparable-projects/>.
- [3] The HDF Group. *HDF-Java 2.10 release*. 2013. URL: <http://www.hdfgroup.org/products/java/hdf-java-html/>.
- [4] Google. *Google Protocol Buffers*. 2012. URL: <https://developers.google.com/protocol-buffers/>.
- [5] Apache Avro Project Members. *Apache Avro 1.7.6*. 2013. URL: <https://avro.apache.org/>.
- [6] Apache Thrift Project Members. *Apache Thrift 0.9.1*. 2013. URL: <https://thrift.apache.org/>.
- [7] Apache Hadoop Project Members. *Apache Hadoop 2.2.0*. 2013. URL: <https://hadoop.apache.org/>.
- [8] Storm Development Team. *STORM: Distributed and fault-tolerant realtime computation*. 2013. URL: <http://storm-project.net/>.
- [9] Sadayuki Furuhashi. *MessagePack - It's like JSON but fast and small*. 2013. URL: <http://msgpack.org/>.
- [10] R. Fielding et al. *Hypertext Transfer Protocol - HTTP/1.1*. 1999. URL: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.
- [11] JSON. *JavaScript Object Notation*. 2013. URL: <http://www.json.org/>.
- [12] MongoDB Development Team. *BSON Specification*. 2013. URL: <http://bsonspec.org>.

- 
- [13] Inc. MongoDB. *MongoDB Manual, Version 2.4*. 2014. URL: <http://docs.mongodb.org/manual/>.
- [14] Michael Tiller. *recon – Web and network friendly simulation data formats, Python Interface*. 2013. URL: <https://github.com/xogeny/recon>.
- [15] Jackson JSON processor project team. *Smile Format Specification*. 2013. URL: <http://wiki.fasterxml.com/SmileFormatSpec>.
- [16] Pietrzak Roman. *Binary-JSON (BJSON) format specification*. 2013. URL: <http://bjson.org/>.
- [17] Riyad Kalla. *Universal Binary JSON Specification*. 2013. URL: <http://ubjson.org>.
- [18] Julian Seward. *Bzip2 Homepage*. 2010. URL: <http://www.bzip.org/>.
- [19] Peter Harman. *recon – Web and network friendly simulation data formats, C Interface*. 2014. URL: <https://github.com/harmanpa/crecon>.
- [20] Peter Harman. *jrecon – Web and network friendly simulation data formats, Java Interface*. 2014. URL: <https://github.com/harmanpa/jrecon>.



# IDOS - (also) a Web Based Tool for Calibrating Modelica Models\*

Radoslaw Pytlak    Tomasz Tarnawski

Warsaw Technical University, Institute of Automatic Control and Robotics  
ul. Sw Andrzeja Boboli 8, 02-525 Warsaw

## Abstract

This paper presents a newly deployed server, IDOS, an online-accessible environment providing the service of solving optimal control problems. Development and deployment of the Interactive Dynamic Optimization Server is a result of projects funded by NCBiR (National Center for Research and Development). One of the outcomes of the project was a modeling language (Dynamic Optimization Modeling Language, DOML) providing a uniform format for defining dynamic optimization problems. DOML is an extension of Modelica language and hence, not only a user can specify his problem in the way he does in Modelica but also (more importantly, for the purpose of this paper) models created in Modelica for simulation purposes can be easily transferred to DOML for solving their related optimization problems. In particular, Modelica models can be calibrated with the help of our server. The paper tries to illustrate the point in depth. It presents the workings of the server and reviews the scope of solvers implemented, focusing especially on those that can be used for calibrating Modelica models. Special attention is devoted to an algorithm using adjoint equations for evaluating sensitivities of model equations with respect to parameters and to calibrating models described by higher index DAEs.

*Keywords: dynamic optimization; optimal control; model calibration*

## 1 Introduction

The paper describes computing environment IDOS accessed by Internet and created for solving opti-

mal control problems. The IDOS server is equipped with its Dynamic Optimization Modeling Language (DOML) for defining optimal control problems. The DOML format provides a mean for describing an optimal control problem in slightly extended Modelica syntax. It is very similar to Optimica ([1],[2]) and in fact its implementation is based on the JModelica.org Optimica compiler, although a number of features proposed in DOML set it apart from Optimica (these are described in detail in the accompanying paper [24]).

The IDOS server was created to provide means for solving optimal control problems by essentially different solvers such as: multiple shooting methods; methods based on *a priori* discretization of systems equations and then transforming the problem to a large-scale nonlinear programming problems; methods which use adjoint equations in order to evaluate reduced gradients of functionals defining the problem. From the beginning we have followed the idea that the server should enable a strategy of chaining of solvers—starting with some solver to get a rough approximation to the optimal control solution and then proceeding further with another solver which completes the solution process by giving an accurate solution to the problem. Having this in mind we knew from the beginning that the server should have as many different solvers as possible. Since all these solvers were to be invoked from the DOML script this approach enabled us to provide quite precise requirements for the dynamic optimization language. In the paper [24] we describe how that approach has been materialized in the form of new constructs put into the Modelica framework.

In the following sections the paper briefly describes the IDOS server by paying much attention to the types of optimal control problems which are supported by the server. Then we describe in some details numerical procedures which are avail-

\*The work presented in the paper has been partially supported by NCBiR grants: R02-0009-06, PBS1/A7/6/2012

able at the IDOS server and which can be used to calibrate various Modelica type problems. These solvers constitute a subset of all solvers available at IDOS which are also mentioned in the paper.

## 2 The IDOS server

The main task of the IDOS platform ([20],[19],[21],[26]) is to manage the processes of solving the dynamic optimization problems. Number of tasks that can be solved over a period of time depends on the server CPU. When its computing power is insufficient, optimization task are queued to be executed when the server resources are released. The solving process on IDOS is build of the general steps: 1) define the optimization problem using dedicated language (DOML, Dynamic Optimization Modeling Language); 2) submit the problem to IDOS platform; 3) interpret obtained results.

During the design process of the IDOS platform the following assumptions were made: 1) the primary goal of the system is to allow users to define, calculate and view results of optimization tasks; 2) all functions of the system will be available remotely via the Internet, without the need to install dedicated software on the user's machine. Different tools used to developed web applications will be used to fulfill this task; 3) the system is designed to manage several types of optimization libraries; 4) the system design is modular so it can be expanded and launched in stages (for example we can extend the system by adding computation servers or implementing support for newly developed numeric libraries); 5) the system should include user management module (registration, login). Each user should be able to define, calculate their own optimization tasks (within own account on the IDOS platform); 6) the results of optimization calculations shall be presented in a universal, unified form.

Each submitted optimization problem is processed in a pipeline fashion, beginning with a task data collection from a database and ending with storing the results. After a task is stored in a database and its status set as ready problem is located by a Data Base Adapter daemon (a process continuously running on the server). Next prepared task is analyzed by the DOML Compiler and in result C++ source code is generated (or compilation errors are reported if the submitted definition

was faulty). In the following step a C++ code is compiled. The final binary file is built on the base of optimization libraries stored on the IDOS server. The last step is the execution of the build binaries. When the job is finished, results are stored in the Central Archive database. If some error appeared its description is stored in the Central Configuration Database and the status of a task is set as failed. The optimization results are stored in uniquely named XML files. The name and location of each such file is stored in a central database.

## 3 Solvers implemented and libraries used

The IDOS server has been built to solve primarily the following optimal control problem

$$\min_u \phi(x(t_f)) \tag{1}$$

subject to the constraints:

$$F(\dot{x}, x, u, t) = 0 \text{ a.e. } t \in T = [0, t_f] \tag{2}$$

$$q(x(t), t) \leq 0, t \in T \tag{3}$$

$$h_i^1(x(t_k)) = 0, i \in E \tag{4}$$

$$h_j^2(x(t_k)) \leq 0, j \in I \tag{5}$$

$$u(t) \in \Omega \text{ a.e. } t \in T. \tag{6}$$

In general we assume that the structural index of equations (2) does not exceed three ([16]). If differential–algebraic equations in the above problem reduce to ODEs then more general problems (with respect to constraints) can be handled by the server.

At the moment the IDOS server can handle control problems described with ordinary equations and differential–algebraic equations but the incorporation into the IDOS solvers for problems with partial differential equations is also under way.

The IDOS server enables solving optimal control problems by using essentially different methods:

- a) based on an *a priori* discretization of systems equations–optimal control problem is then transformed to a large scale nonlinear programming problem;
- b) based on numerical methods for integrating system equations with variable stepsizes–reduced gradients are then evaluated with the help of the corresponding adjoint equations;



- c) based on shooting methods applied to differential equations derived from necessary optimality conditions for optimal control problems;
- d) based on adjoint equations for problems described by higher index differential–algebraic equations;
- e) based on Mixed Integer Programming algorithms for optimal control problems with ordinary differential equations in which some control variables take only integer values.

Obviously, not all solvers can be used on the every problem. Their applicability is briefly discussed below.

### 3.1 Optimal control problems described by ODEs

As far as optimal control problems described by ODEs are concerned essentially three different approaches to solving them are applied.

**In the first approach** system equations are discretized *a priori* (with respect to time) and in effect replaced by nonlinear algebraic equations. As a result, a large scale nonlinear programming problem (NLP) is solved. The main library used in this case is the OLADO package ([3]) which contains a wide range of optimization solvers, mostly based on some version of the SQP algorithm. Furthermore, in the OLADO package three different interior point QP solvers are available: procedure which implements Mehrotra’s primal-dual predictor-corrector method [13]; procedure based on the Mehrotra’s algorithm modified by R. Franke [6]; and the procedure which applies the Gondzio’s multiple centrality correctors variant of the primal-dual interior-point method for convex QP [8].

The IDOS server is also equipped with sparse nonlinear optimization HQP solver and its OMUSES interfaces to ODEs integrators ([7]). In particular, the HQP solver is linked with: procedures employing Euler, or simple Runge–Kutta fourth order rules (RK4); GRK4 procedure of Rosenbrock’s type ([11]); DOPRI5 procedure based on explicit Runge–Kutta scheme due to Dormand and Prince ([10]); IMP procedure which uses the implicit midpoint rule ([11]); SDIRK procedure based on singly diagonally implicit Runge–Kutta formula ([11]) and

with ODETS procedure which uses Taylor’s expansion of ODEs derived from ADOL–C driver routine `forodec`.

**The second approach** ‘preserves’ the continuous time of the optimal control problem to be solved. It means that during numerical treatment the system’s equations are integrated by a procedure with variable stepsizes. Since state variables are not decision variables the problem is considered in the, so called, reduced space, i.e. all functionals defining the problem are functionals of control variables only—gradients of these functionals, required by an optimization procedure, are evaluated with the help of the adjoint equations which are consistent with the system equations.

Essentially, there are two procedures on the IDOS server which follow this approach. In the first one, system equations are integrated by procedures from the `cvodes` part of the SUNDIALS package ([22]), also adjoint equations are integrated by procedures from the `cvodes`. Iteration of the optimization procedure is performed by IPOPT ([25]). In the second procedure, implicit Runge–Kutta method is employed for system equations integration and adjoint equations are evaluated in accordance with the discretization of ODEs by the integration procedure. In this case optimization is performed by the SQP method which applies an active set strategy ([17]). The procedure described in [17] was designed to cope efficiently with control problems with state constraints.

**The third approach** to optimal control problems with ODEs deals with multiple shooting methods. At the moment within the IDOS server an indirect shooting method based on Oberle’s code ([14]) is implemented.

### 3.2 Optimal control problems described by DAEs

The server currently offers two procedures for solving optimal control problems with DAEs. The first one is aimed at problems whose differential–algebraic equations have index one. It is based on the implicit Runge–Kutta integration procedure but adopted to index one DAEs in the semi-explicit form ([17]). The optimization engine used here is the SQP procedure which uses an active set strategy in corresponding QP subproblems (a range–space variant of it).

The second procedure carries out the unique approach to optimal control problems with higher

index DAEs described in [18]. The approach does not require the system to be transformed from DAEs to ODEs (through differentiation, with respect to time, of selected algebraic equations). Instead, the constructed procedure employs an implicit Runge–Kutta method for system equations integration. We use Radau IIA method as implemented in [11] (see also [9]). Reduced gradients are evaluated on the basis of adjoint equations defined for discretized (by the integration procedure) system equations. The procedure uses the algorithm for consistent initialization of system equations and for that purpose the method based the Pantelides’ algorithm ([15]) and the `kinsol` procedure from the SUNDIALS package was implemented. To our knowledge, it is the first successfully implemented procedure for solving optimal control problems described by higher index DAEs which does not need transformation of the DAEs to the underlying ODEs.

### 3.3 Optimal control problems with integer valued controls

For the moment, optimal control problems with integer valued controls solved at the IDOS server can only have ordinary differential equations. For these problems we used BONMIN package (see e.g. [4], [5]). It has been integrated with the OLADO library and with the procedures based on the `cvodes` procedures from SUNDIALS package.

Table 1 summarizes to some extent the solvers structure of the IDOS server — in fact it is much more complex since, for instance, procedures such as `cvodes`, or BONMIN contain themselves a number of essentially different methods which can be accessed by setting their parameters. Furthermore, the server employs ADOL–C package for performing automatic differentiation and OpenBLAS package for linear algebra operations.

The code developed for the server is meant as open source<sup>1</sup> (most of the libraries used by the server are open-source, as well, see Table 1), hence one could deploy own instances of IDOS server, at least in principle. The experience teaches, however, that installation, configuration and maintenance of all required packages is in itself far from trivial. For that reason, at this point there is no dedicated distribution package for automated in-

<sup>1</sup>contact the authors for setting up access to code repository

Method and problem type	Solvers
<i>a priori</i> discretization, ODEs	OLADO: SQP (Powell’s type) + Euler IPOPT + Euler HQP + Euler HQP + OMUSES: HQP + Euler HQP + DOPRI5 HQP + GRK4 HQP + IMP HQP + ODETS HQP + SDIRK HQP + RK4
adjoint equations, ODEs	RKCON + Radau IIA IPOPT + SUNDIALS ( <code>cvodes</code> )
shooting method, ODE	BNDSCO + RK4
<i>a priori</i> discretization, ODEs (integer controls)	BONMIN + Euler SQP (Powell’s type) + Euler
adjoint equations, ODE (integer controls)	BONMIN + SUNDIALS ( <code>cvodes</code> )
adjoint equations, DAEs (index = 1)	RKCON + Radau IIA IPOPT + SUNDIALS ( <code>idas</code> ) - u. dev.
adjoint equations, DAEs (index ≤ 3)	RKCON + RADAU5 + SUNDIALS ( <code>kinsol</code> ) + MAXIMA

Table 1: Listing of IDOS main solvers

stallation of the server suite on user machines, although development of such is planned.

## 4 Estimation of models parameters

Suppose that we have the dynamic model with parameters  $p$ :

$$\dot{x}(t) = f(x(t), t, p), \quad t \in [0, t_f], \quad x(0) = x_0. \quad (7)$$

which solution  $x^p$  is dependent on parameters  $p$ . It is often the case that such model describes a real life, observable phenomenon but the exact values of its parameters are unknown. Then, they could be approximated by solving nonlinear least squares problem:

$$\min_p \sum_{k=1}^N [(x_l(t_k) - x_l^e(t_k))^2] \quad (8)$$

subject to the constraints (7), where  $x_l^e$  is an empirical (measured) trajectory.

Since  $x$  is a function of parameters  $p$ , the problem can be stated as

$$\min_p \left[ F(p) = \sum_{k=1}^N (x_l^p(t_k) - x_l^e(t_k))^2 \right] \quad (9)$$

That problem can be solved by nonlinear programming techniques provided that we can evaluate gradients of  $F(p)$ . In particular, we can apply the Gauss–Newton algorithm (see section 5) to problem (9) taking advantage of the least–squares structure of its objective function.

The gradient of the functional  $F(p)$  may be evaluated with the help of adjoint equations if we observe that  $F(p)$  is composed of  $N$  functions  $F^k(p)$ :

$$F^k(p) = (x_l^p(t_k) - x_l^e(t_k))^2$$

where  $x^p$  is the solution to the equations (7).

The gradient of  $F^k(p)$  is given by the formula:

$$\nabla F^k(p) = \int_0^{t_k} f_p(x(t), t, p)^T q_k(t) dt \quad (10)$$

where  $q_k$  is the solution to the adjoint equations

$$\begin{aligned} q_k(t_k) &= Q_k \\ \dot{q}_k(t) &= -f_x(x(t), t, p)^T q_k(t), \quad t \in [0, t_k). \end{aligned}$$

Here,

$$Q_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 2(x_l(t_k) - x_l^e(t_k)) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Then  $\nabla F(p) = \sum_{k=1}^N \nabla F^k(p)$ .

Having objective function values and its gradients we can build an optimization procedure for model calibration. A general scheme for these procedure may look like stated below:

### General Calibration Procedure

1. Set initial values of parameters:  $p_1$  and set  $k = 1$ .
2. For parameters  $p_k$  calculate system trajectories  $x^{p_k}$  by numerically integrating system equations using, for example, procedure `cvoides` from SUNDIALS package. On that basis determine objective function value  $F(p_k)$  through values  $F^l(p_k)$ ,  $l = 1, \dots, N$ .
3. Having trajectories  $x^{p_k}$  and values  $F^l(p_k)$ ,  $l = 1, \dots, N$  solve adjoint equations (using, for example procedure `cvoides`), and determine  $\nabla F^l(p_k)$ ,  $l = 1, \dots, N$ s and  $\nabla F(p_k)$ .

4. Determine the direction of descent using some optimization procedure (for example, `Ipopt`).
5. Perform directional minimization with the help of optimization procedure to evaluate step size  $\alpha_k$ . Substitute  $p_{k+1} = p_k + \alpha_k d_k$ . increase  $k$  by one and go to Step 2).

## 5 Gauss–Newton method for the least squares problem

One approach to implementing the just-outlined general calibration procedure makes use of the Gauss–Newton method for solving nonlinear least squares problems. For the purpose of discussing it, consider the following optimization problem:

$$\min_{x \in \mathcal{R}^n} \left[ f(x) = \frac{1}{2} \|g(x)\|^2 = \frac{1}{2} \sum_{i=1}^m \|g_i(x)\|^2 \right] \quad (11)$$

where

$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{bmatrix}. \quad (12)$$

In this case, the gradient and the Hessian matrix of the objective function (11) are defined as follows:

$$\begin{aligned} \nabla f(x) &= \sum_{i=1}^m g_i(x) \nabla g_i(x) = J(x)^T g(x) \\ \nabla^2 f(x) &= \sum_{i=1}^m \nabla g_i(x) \nabla g_i(x) + \sum_{i=1}^m g_i(x) \nabla^2 g_i(x) \\ &= J(x)^T J(x) + \sum_{i=1}^m g_i(x) \nabla^2 g_i(x). \end{aligned}$$

Here,  $J(x)$  is the Jacobian matrix of the transformation matrix (12). Assuming that  $g_i(x) \nabla^2 g_i(x) \approx 0$  we can write

$$\nabla^2 f(x) \approx J(x)^T J(x).$$

The Gauss–Newton method minimizes, at every iteration, linearization of  $g$  at the point  $x_k$ :

$$\tilde{g}(x_k; x) = g(x_k) + J(x_k)(x - x_k).$$

The next point  $x_{k+1}$  is obtained by solving the problem

$$\min_{x \in \mathcal{R}^n} \frac{1}{2} \|\tilde{g}(x_k; x)\|^2.$$

Then

$$x_{k+1} = x_k - \left( J(x_k)^T J(x_k) \right)^{-1} J(x_k)^T g(x_k) \quad (13)$$

It is a descent method since

$$-J(x_k)^T g(x_k)$$

is a descent direction for the function  $\frac{1}{2} \|g(x_k)\|^2$ , because

$$\nabla \left( \frac{1}{2} \|g(x_k)\|^2 \right) = J(x_k)^T g(x_k)$$

and provided that

$$\left( J(x_k)^T J(x_k) \right) > 0$$

(it means matrix positiveness).

The outlined algorithm is most often applied with the modifications

$$x_{k+1} = x_k - \alpha_k \left( J(x_k)^T J(x_k) + \Delta_k \right)^T \times J(x_k)^T g(x_k), \quad (14)$$

where  $\alpha_k > 0$  is the result of the directional minimization, and  $\Delta_k$  is such diagonal matrix that

$$\left( J(x_k)^T J(x_k) + \Delta_k \right) > 0.$$

If the least squares problem is used to calibrate a model then on the basis of experimental data  $\{y_i, z_i\}_1^m$  we evaluate models parameters  $x$

$$z = h(y, x).$$

In this case models parameters are calculated by solving the optimization problem

$$\min_x \left[ f(x) = \frac{1}{2} \sum_{i=1}^m \|z_i - h(y_i, x)\|^2 \right].$$

The Gauss–Newton approach was implemented along the lines of the *General Calibration Procedure* presented in Section 4. In Step 4) of the procedure, the direction of descent  $d_k$  is determined

on the basis of vectors  $\nabla F^l(p_k)$ ,  $l = 1, \dots, N$  and  $\nabla F(p_k)$ . By referring to relations (13)–(14) we set

$$\begin{aligned} J(p_k) &= \begin{bmatrix} \nabla F^1(p_k) \\ \nabla F^2(p_k) \\ \vdots \\ \nabla F^N(p_k) \end{bmatrix} \\ H(p_k) &= J(p_k)^T J(p_k) \\ p_{k+1} &= p_k - \alpha_k [H(p_k)]^{-1} \nabla F(p_k). \end{aligned} \quad (15)$$

The formula (15) uses the stepsize  $\alpha_k$  which usually is evaluated with the help of sophisticated procedures. We decided not to build new optimization packages but to adopt existing ones therefore  $\alpha_k$  is determined according to Ipopt or RKNCON procedures. Similarly, scaling matrices  $H(p_k)$  are made nonsingular by procedures built-in Ipopt or RKNCON. For example, it can be done in Ipopt by ‘cheating’ the method eval-h with matrices  $H(p_k)$  instead of true Hessian matrices evaluated at  $p_k$ .

## 6 Estimation of parameters of higher index DAEs

The estimation procedure stated in Section 4 can also be adopted to higher index DAEs provided that numerical procedure presented in [18] is used. Suppose that our system equations are as follows:

$$F(\dot{x}(t), x(t), t, p), \quad t \in [0, t_f] \quad (16)$$

and, as before,  $p \in \mathcal{R}^m$  is the vector of unknown parameters.

When the integration scheme from [11] is extended to implicit equations then we will arrive at equations

$$F(x'_i(k+1), x(k) + h(k) \sum_{j=1}^s a_{ij} x'_j(k+1), t_k + c_i h(k), p) = 0, \quad (17)$$

$$x(k+1) - x(k) + h(k) \sum_{i=1}^s b_i x'_i(k+1) = 0, \quad (18)$$

$i = 1, \dots, s$  and they represent the dynamics of a discrete time control system.

In order to define state equations we introduce the state vector  $X(k)$ :

$$X(k) = \begin{bmatrix} x'_1(k) \\ \vdots \\ x'_s(k) \\ x(k) \end{bmatrix}, \quad (19)$$

then equations (17)–(18) become

$$\tilde{F}(X(k+1), X(k), k, p) = 0. \quad (20)$$

Here,  $X(k) \in \mathcal{R}^{(s+1)n}$ ,  $\tilde{F} : \mathcal{R}^{(s+1)n} \times \mathcal{R}^{(s+1)n} \times \{0, \dots, N-1\} \times \mathcal{R}^m \rightarrow \mathcal{R}^{(s+1)n}$ .

System (20) is fully implicit discrete time and, under some nonsingularity assumption, can be expressed as explicit. If the Jacobian of  $\hat{F}$  with respect to  $X(k+1)$ , denoted by  $\tilde{F}_{X+}$ , exists and is nonsingular for all  $k = 0, \dots, N-1$ , then from the Implicit Function Theorem there exists unique function  $\varphi$  such that

$$X(k+1) = \varphi(X(k), k, p) \quad (21)$$

and

$$\tilde{F}(\varphi(X(k), k, p), X(k), k, p) = 0, \quad (22)$$

for  $k = 0, \dots, N-1$ . Under differentiability assumptions imposed on  $F$  the function  $\varphi$  is differentiable with respect to  $X(k)$  and  $p$  which means that we can write

$$\begin{aligned} \tilde{F}_{X+}(k)\varphi_X(k) + \tilde{F}_X(k) &= 0 \Rightarrow \\ \varphi_X(k) &= -[\tilde{F}_{X+}(k)]^{-1} \tilde{F}_X(k) \end{aligned} \quad (23)$$

$$\begin{aligned} \tilde{F}_{X+}(k)\varphi_p(k) + \tilde{F}_p(k) &= 0 \Rightarrow \\ \varphi_p(k) &= -[\tilde{F}_{X+}(k)]^{-1} \tilde{F}_p(k). \end{aligned} \quad (24)$$

$\tilde{F}_{X+}(k)$ ,  $\tilde{F}_X(k)$ ,  $\tilde{F}_p(k)$  are evaluated at  $(X(k+1), X(k), k, p)$  and  $\varphi_X(k)$ ,  $\varphi_p$  at  $(X(k), k, p)$ .

If we consider optimal control problem with the objective function

$$\hat{F}^0(p) = (x_l^p(t) - x_l^e(t))^2 \quad (25)$$

then the gradient of it can be calculated by referring to adjoint equations for the functional (25) and the system (21).

The adjoint equations for the functional (25) and the system (21) are considered, for example, in [17]:

$$p(t) = Q_t \quad (26)$$

$$p(k) = \varphi_X(k)^T p(k+1), \quad (27)$$

$k = 0, \dots, t-1$  and

$$Q_t = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 2(x_l^p(t) - x_l^e(t)) \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Notice that vector  $Q_t$  has dimension  $(n+1) \times s$ .

Then adjoint variables  $p$  are the means for the gradient evaluation according to the formula

$$\hat{F}_p^0(p) = \sum_{k=1}^{t-1} \varphi_p(k)^T p(k+1). \quad (28)$$

Using (23)–(24) the adjoint equations (27) and the formula (28) can be expressed without the knowledge of  $\varphi$ :

$$\begin{aligned} p(k) &= -\tilde{F}_X(k)^T [\tilde{F}_{X+}(k)]^{-T} p(k+1) \\ \hat{F}_p^0(p) &= -\sum_{k=1}^{s-1} \tilde{F}_p(k)^T [\tilde{F}_{X+}(k)]^{-T} p(k+1). \end{aligned}$$

Fortunately, the calculation of  $[\tilde{F}_{X+}(k)]^{-1}$  can be avoided with the help of additional variable  $r$  which is the solution to the linear equations

$$\tilde{F}_{X+}(k)^T r(k+1) = p(k+1). \quad (29)$$

Then the adjoint equations become

$$p(k) = -\tilde{F}_X(k)^T r(k+1) \quad (30)$$

$$\hat{F}_p^0(p) = -\sum_{k=1}^{s-1} \tilde{F}_p(k)^T r(k+1). \quad (31)$$

Eventually we have the viable formula for the gradient of  $\hat{F}(p)$  provided that matrices  $\tilde{F}_{X+}(k)$  are nonsingular and equations (29) can be solved efficiently.

Important observation is that matrices  $\tilde{F}_{X+}(k)$  are **nonsingular** even for index three problems provided that  $h(k)$  are sufficiently small—see [18] for details. Therefore, for many higher index DAEs we have a valid technique for computing gradients of  $\hat{F}^0(p)$ . The evaluation of adjoint equations requires solving linear equations with matrices which have to be evaluated (and factorized) anyway while numerically integrating system equations.

The estimation procedure discussed in this section requires the program for consistent initialization at initial time. That program consists of two steps.

### Consistent Initialization Procedure

1. In the first step new equations have to be determined by differentiating some equations with respect to time.
2. In the second step the extended set of equations is solved, at initial time, to get initial values of original equations.

In our approach Step 1) of *Consistent Initialization Procedure* is carried out on the basis of a graph based algorithm proposed by Pantelides ([15]). The algorithm finds the minimal subset of equations which need to be differentiated.

Next symbolic differentiation is performed. A good enough tool with symbolic differentiation module is an open source computer algebra system Maxima ([23]) which is descendant of Macsyma, the computer algebra system developed in the late 1960s at MIT. In the next section we show that we start solving optimal control problems by stating them in the DOML environment. In that case the first step in the symbolic differentiation is a conversion of the equations defined in the DOML format to the Maxima form. Then, the Maxima library is invoked and as a result differentiated equations are returned to the system console.

Eventually, in Step 2) of *Consistent Initialization Procedure*, the set of nonlinear (in general) algebraic equations is solved by using the KINSOL solver from the SUNDIALS package ([12]) (alternatively the Ipopt solver can be applied-[25]).

Summing up, the IDOS server is equipped, for the moment, with optimization procedures which either use quasi-Newton approximations of Hessian matrices (if Ipopt or RKNON are used without modifications), or their Gauss-Newton approximations. They are based on adjoint equation evaluations and can be applied to models described by ODEs or higher index DAEs. Procedures for models calibration based on *a priori* discretization of systems equations are under development.

## 7 Examples

**Example 1** The first example is the chemical kinetics model available as SUNDIALS example. The equations of the reaction rates are as follows:

$$\begin{aligned} \dot{x}_1 &= -p_1x_1 + p_2x_2x_3 \\ \dot{x}_2 &= p_1x_1 - p_2x_2x_3 - p_3x_2^2 \\ \dot{x}_3 &= p_3x_3^2 \end{aligned}$$

The models transcription in Modelica is shown on Listing 1. The corresponding DOML file for the model calibration is presented on Listing 2. It defines the reference trajectory through the spline function of degree 1. All three parameters there are decision variables.

When  $p_1 = 0.04$ ,  $p_2 = 1.e7$  and  $p_3 = 3e7$  we have the SUNDIALS example and we call the model with

```

package Roberts
  model Roberts_model(startTime=0.0,
                      finalTime=1.0 )
    parameter Real p1 = 0.04;
    parameter Real p2 = 1e4;
    parameter Real p3 = 3e7;
    Real x1(start = 1.0);
    Real x2(start = 0.0);
    Real x3(start = 0.0);
  equation
    der(x1) + p1*x1 - p2*x2*x3 = 0;
    der(x2) - p1*x1 + p2*x2*x3 + p3*x2*x2 = 0;
    der(x3) - p3*x2*x2 = 0;
  end Roberts_model;
end Roberts;

```

Listing 1: Roberts model in Modelica

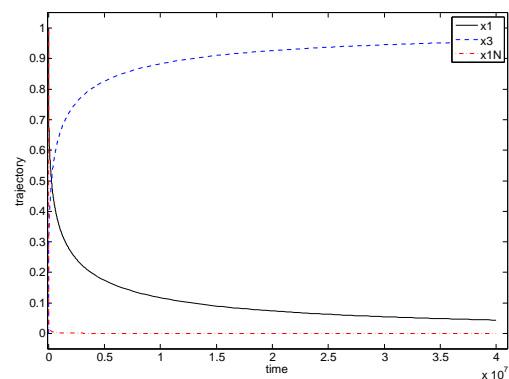


Figure 1: The calibration of Roberts model.

```

package Roberts
  import I = Modelica.DOML.Inputs;
  optimization Roberts_opt(startTime=0.0,
                           finalTime=1.0 )
    minimize AccErr = err(finalTime);
    I.Spline x1r(startTime = startTime,
                 finalTime = finalTime, degree=1,
                 table=[0.0, 0.6;8e6, 0.3; 16e6, 0.2;
                        24e6,0.1;32e6,0.01;40e6,0.0;4.0e7,0.0]);
    parameter Real p1(free=true,
                      initialGuess=0.04, min=0.0, max=10.0);
    parameter Real p2(free=true,
                      initialGuess=1e4, min=0.0, max=10.0);
    parameter Real p3(free=true,
                      initialGuess=3e7, min=0.0, max=10.0);
    input Real x1e(free=false) = x1r.y;
    Real x1(start = 1.0);
    Real x2(start = 0.0);
    Real x3(start = 0.0);
    Real err(start = 0.0);

    annotation(solver="nlsq_cvodes", Steps="5",
              RTOL_tolerance="1.0e-6");
  equation
    der(x1) + p1*x1 - p2*x2*x3 = 0;
    der(x2) - p1*x1 + p2*x2*x3 + p3*x2*x2 = 0;
    der(x3) - p3*x2*x2 = 0;
    der(err) = (x1-x1e)^2;
  end Roberts_opt;
end Roberts;

```

Listing 2: Calibrating Roberts model

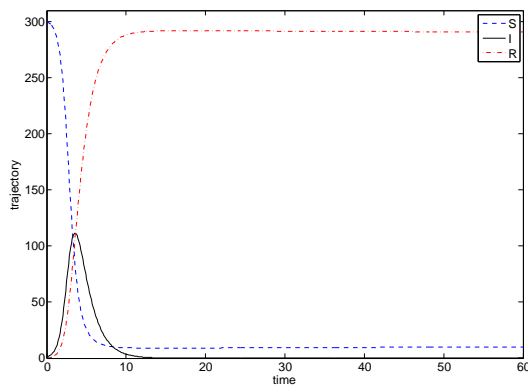


Figure 2: The calibration of the extended SIR model.

these parameters as the nominal model (its trajectories are denoted as  $x_N$  in Figure 1). However, when the model was adjusted to the reference trajectory for  $x_1$ :  $(0.0, 0.6)$ ,  $(8e6, 0.3)$ ,  $(16e6, 0.2)$ ,  $(24e6, 0.1)$ ,  $(32e6, 0.01)$ ,  $(40e6, 0.0)$ ,  $(4.0e7, 0.0)$  models parameters were changed to  $p_1 = 1e - 3$  (we assumed that lower bound for the parameter  $p_1$  is  $1e - 3$ ),  $p_2 = 2.1986e4$  and  $p_3$  to  $3.0e7$ . Also trajectories of the model changed—they are shown on Fig. 1.

**Example 2** The second example is related to the model of food borne diseases. The equations of the evolution of three populations:  $S$  (Susceptible population),  $I$  (Infected population),  $R$  (Recovered population) and the equation of the pathogen concentration evolution are given below.

$$\begin{aligned}\dot{S} &= -p_3(B/(B+p_2))S - ((p_4p_5)/p_8)SI - \\ &\quad p_1S + p_1p_8 \\ \dot{I} &= p_3(B/(B+p_2))S + ((p_4p_5)/p_8)SI - p_1I \\ \dot{R} &= p_1I - p_6R \\ \dot{B} &= p_7I - p_5B.\end{aligned}$$

The model has eight parameters, all of them are decision variables in our calibration problem. In addition we assume that initial pathogen concentration,  $B(0)$ , must be determined. The trajectories of the calibrated model are shown on Fig. 2.

## 8 Conclusions

The IDOS server is still a prototype. Although it is operational its widespread use can be hampered by the limited computing resources which are currently allocated to the server. Furthermore, adding a new solver to the server is not as simple as it should be. We plan to alleviate these problems in the near future.

## References

- [1] Åkesson J. Tools and Languages for Optimization of Large-Scale Systems. Lund, Sweden: *PhD thesis*, Department of Automatic Control, Lund University, 2007.
- [2] Åkesson J. Optimica—an extension of Modelica supporting dynamic optimization. In: Proceedings of the 6th Modelica Conference 2008, Bielefeld, Germany, Modelica Association, 3-4 March 2008.
- [3] Błaszczuk J, Karbowski A, K. Malinowski K. Object Object library of algorithms for dynamic optimization problems: benchmarking SQP and nonlinear interior point methods. In: Journal of Applied Mathematics and Computer Science, Vol. 17, 515–537, 2007.
- [4] Bonami P, Lee J BONMIN Users' Manual, 2009.
- [5] Bonami P, Lodi A, Cornujols G, Margot F. A feasibility pump for mixed integer nonlinear programs. In: Mathematical Programming, Vol. 119, 331–352, 2009.
- [6] Franke R. Anwendung von Interior-Point-Methoden zur Lösung zeitdiskreter Optimalsteuerungsprobleme. *Master's thesis*, Technische Universität Ilmenau, Institut für Automatisierungs- und Systemtechnik Fachgebiet Dynamik und Simulation ökologischer Systeme, Ilmenau, Germany, October 1994 (in German).
- [7] Franke R. OMUSES — a Tool for the Optimization of MULTistage Systems and HQP — a Solver for Sparse Nonlinear Optimization. Dept. of Automation and Systems Engineering, Technical University of Ilmenau, Germany, September 1998.

- [8] Gondzio J. Multiple centrality corrections in a primal-dual method for linear programming. *Technical Report 1994.20*, Department of Management Studies, University of Geneva, Geneva, Switzerland, 1994.
- [9] Hairer E, Lubich Ch, Roche M. The numerical solution of differential–algebraic equations by Runge–Kutta methods. *Lecture Notes in Mathematics 1409*, Springer–Verlag, Berlin, Heidelberg, 1989.
- [10] Hairer E, Norsett S.P, Wanner G. *Solving Ordinary Differential Equations I.*, Springer–Verlag, Berlin Heidelberg New York, 2008.
- [11] Hairer E, Wanner G. *Solving Ordinary Differential Equations II*, Springer–Verlag, Berlin Heidelberg New York, 1996.
- [12] Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., Woodward, C.S. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, preprint, UCRL-JRNL-200037, 2004.
- [13] Mehrotra S. On the implementation of a primal–dual interior point method. In: *SIAM Journal on Optimization*, Vol. 2, 575–601, 1992.
- [14] Oberle H.J, Grimm W. BNDSCO – A Program for the Numerical Solution of Optimal Control Problems, Report No. 515 der DFVLR (German Test and Research Institute for Aviation and Space Flight), 1989.
- [15] Pantelides C. Consistent initialization of differential–algebraic system. In: *SIAM J. Scientific and Statistical Computation* Vol. 9, 213-231, 1988.
- [16] Pryce, J.D. A simple structural analysis method for DAEs, *BIT* 41 (2001), pp. 364–394.
- [17] Pytlak R. Numerical procedures for optimal control problems with state constraints. *Lecture Notes in Mathematics 1707*, Springer–Verlag, Heidelberg, 1999.
- [18] Pytlak R. Numerical procedure for optimal control of higher index DAEs. In: *J. Discrete and Continuous Dynamical Systems*, Vol. 29, 1–24, 2011.
- [19] Pytlak R, Tarnawski T, Fajdek B, Stachura M. Interactive dynamic optimization server–connecting one modeling language with many solvers. In: *Optimization Methods and Software*, 2013. <http://dx.doi.org/10.1080/10556788.2013.799159>.
- [20] Pytlak R (ed.). *Interactive computer environment for solving optimal control problems–IDOS*. BelStudio, Warsaw, Poland, 2012.
- [21] Pytlak R, Błaszczuk J, Krawczyk K, Tarnawski T. Solvers chaining in the IDOS server for dynamic optimization. In: *Proceedings of 52nd Conference on Decision and Control*, Florence, Italy, 10-13 December 2013.
- [22] Serban R, Hindmarsh A.C. CVODES, the sensitivity-enabled ODE solver in SUNDIALS. In: *Proceedings of the 5th International Conference on Multibody Systems, Nonlinear Dynamics and Control*, ASME, Long Beach, CA, 2005.
- [23] de Souza, N., Fateman, R.J., Moses, J., Yapp, C. *The Maxima Book*, <http://maxima.sourceforge.net/docs/maximabook/maximabook-19-Sept-2004.pdf>, 2004.
- [24] Tarnawski, T, Pytlak R. DOML - a compiler environment for dynamic optimization supporting multiple solvers. Accepted for: 10th Modelica Conference 2014, Lund, Sweden, Modelica Association, 10-12 March 2014.
- [25] Wachter A, Biegler L.T. On the implementation of an interior point Line search filter algorithm for large scale nonlinear programming. In: *Math. Programming*, Vol. 106, 25–57, 2006.
- [26] *Interactive Dynamic Optimization Server*. <http://idos.mchtr.pw.edu.pl/>. November 2013.



# Client-side Modelica powered by Python or JavaScript

Rüdiger Franke, ABB, Germany – Ruediger.Franke@de.abb.com

## Abstract

Modelica is primarily supported by simulation environments for the treatment of equation based models and model libraries. As of today Modelica is rarely used for the exchange of engineering data, visualization or interactive computing, even though the Modelica language offers a lot of interesting features for such applications.

This paper investigates the potential of lightweight Modelica tools that run directly in scripting or web clients. Two Modelica parsers have been implemented in the popular client-side languages Python and JavaScript.

The Modelica parser in Python is extended with a backend translating algorithmic Modelica definitions to Python. This gives access to existing Python packages from scripted Modelica. It also enables the interactive debugging of algorithmic Modelica code.

The Modelica parser in JavaScript offers a generic backend interface. The paper demonstrates two applications. First a simple analysis tool for Modelica packages running from the command line is demonstrated. The true potential of JavaScript is the embedding of engineering data as Modelica code with HTML5 documents and their processing on the client side, e.g. in Web browsers. The paper shows a Modelica text editor and parameter GUI generator running in a web browser.

*Keywords: Modelica, scripting, interactive computing, data exchange, Lex, Yacc, Python, JavaScript, jQuery, HTML5.*

## 1 Introduction

Today's Modelica simulation environments act as servers that offer proprietary client interfaces, basing on Modelica Script, COM, or CORBA, for instance. There has been no success in standardizing Modelica client interfaces so far. XML has been selected for tool coupling in the FMI standard.

Major advantages of XML are that it is both: human and machine readable. XML parsers are readily available. The major drawbacks of XML are its very

basic syntax, making it bulky and hard to read for humans. The semantics still needs to be defined.

This paper investigates the use of Modelica itself as interface language. Modelica is human and machine readable as well. The major advantages of Modelica are its more compact, richer syntax. The semantics is already standardized, tailored for modeling and simulation, including:

- rich syntax for high-level definitions, like packages, classes, records, enumerations, doc strings, and physical units;
- modification syntax for predefined classes that is comparable to XML documents for XML schemata;
- convenient formulation of matrix expressions, statements and functions;
- embedded graphical representation;
- embedded HTML documentation;
- embedded version management;
- enable automatic generation of graphical user interfaces out of Modelica definitions.

The features are unique in their combination and could directly be exploited without the need to define some new XML schema first.

The drawback is that parsing Modelica by machines is not as simple as parsing XML. But a Modelica parser neither is a miracle since the concrete syntax is specified and tools like Lex and Yacc exist.

## 2 MoiPy – Modelica in Python

MoiPy is intended to bridge the gap between the powerful Modelica language and convenient scripting. This is done by adding a thin syntactic layer on top of the existing scripting language Python, translating between Modelica and Python, and by using the NumPy package for scientific computing.

MoiPy is not an alternative to other Modelica tools; it is an optional addition. MoiPy allows staying in the Modelica world even if the simulation environment at hand has only limited support for model-based applications or scripting.

## 2.1 Implementation overview

MoiPy implements the Modelica syntax (Modelica 3.3 specification, Appendix B; see [1]) using Python Lex-Yacc. PLY provides Lex and Yacc entirely in Python, including extensive error checking and logging. The syntax specification can directly be executed. A parser table is generated on demand and cached in the background.

Of course the execution speed is slower, compared to a parser explicitly generated and readily compiled. This is why MoiPy only reads Modelica files as needed.

The parser produces an abstract syntax tree (AST). Each node of the tree is a Python object. The AST reduces the concrete syntax for simpler further processing. For instance, a `class_definition` is represented as object of `ClassDefinition`. The attributes of `class_prefixes`, `class_specifier` and `composition` appear directly in the `ClassDefinition`. The elements and sections of the composition are further reduced into one `elementList`, one `initialEquationList` / `initialStatementList` and one `equationList` / `statementList`.

An exemplary rule reads:

```
def p_annotation(p):
    ''' annotation : ANNOTATION \
        class_modification '''
    p[0] = Annotation(
        classModification = p[2],
        track = Track(p, 1))
```

The syntax rule is specified in the documentation string of a Python function that implements the respective production. A new object of the class `Annotation` is created in the example. The second argument tracks the location in the Modelica code.

The class definition objects have the common method `toPython` that generates Python code from the Modelica definitions. The `toPython` method of `PrimaryUnsignedNumber`, for instance, constructs a predefined `Real` or `Integer` object that adds attributes like `min`, `max` and `unit` to the value itself. Note that the Python code generation only covers the algorithmic part of the Modelica language, i.e. a subset of all possible class definitions:

- Modelica packages are treated as Python modules
- Modelica functions are translated to Python functions
- Modelica records are translated to Python classes

- Modelica enumerations are translated to Python objects

Moreover expressions and statements are covered:

- Modelica expressions are evaluated as Python expressions
- Modelica arrays are treated as NumPy arrays
- Modelica builtin functions are forwarded to Python functions
- Modelica statements are executed as Python statements

## 2.2 Extended Modelica syntax for scripting

MoiPy attempts to stay as close as possible to Modelica, using the same rules as specified in the Modelica concrete syntax and without introducing any new keywords. Two extensions are needed for scripting though: support for commands and use of variables without declaration on the top-level scope.

The Modelica concrete syntax covers stored definitions in the form of class definitions. MoiPy additionally accepts commands on the top-level scope that are Modelica expressions, statements or import clauses.

In Modelica each variable must be declared before use. MoiPy follows this rule inside class definitions. Also on the top-level scope assignments like:

```
v := {1,2,3}
```

are declaration errors if `v` has not been declared before. On the top-level scope, outside class definitions, it is allowed though to implicitly declare a variable by defining it equal to an existing object. For instance:

```
v = {1,2,3}
```

defines `v` to be the vector `{1, 2, 3}`. Afterwards

```
v := {4,5,6}
```

assigns a new value to the existing vector `v`,

```
v := "a string"
```

is an error, whereas

```
v = "a string"
```

(re)defines `v` to be a string.

MoiPy uses the file extension `.moi` to distinguish interpreted script files from regular Modelica definitions.

## 2.3 Basic uses

Start MoiPy with

```
python moi.py
```

A parser table for Modelica is generated when called the first time. Afterwards the `moi>>` command prompt appears.

Do some matrix calculations, e.g.:

```
moi>> A = [1,2;3,4]
moi>> b = {1,1}
moi>> A*b
```

or matrix concatenations:

```
moi>> [A,b]
moi>> [A; transpose([b])]
```

Call a Modelica function from Python:

```
moi>> t = 0:0.01:1
moi>> y = Modelica.Math.sin(
...     2*Modelica.Constants.pi*t)
```

This will look for the Modelica definitions under `MOIPYPATH`, load the files `Modelica/Math/package.mo` and `Modelica/Constants.mo`, translate the function `sin` and the constant `pi` to Python, call the function, and assign the result to the vector `y`. Note that the function call is vectorized automatically.

## 2.4 Enhance Modelica with Python

Python offers many interesting modules, such as Numpy, SciPy and matplotlib. MoiPy enables to access them from Modelica. Type:

```
moi>> import Plt = matplotlib.pyplot
```

MoiPy attempts to find a Modelica definition first. As `matplotlib` is not found in Modelica, the import gets forwarded to Python and found there, provided you have `matplotlib` installed. Type:

```
moi>> Plt.plot(t, y, "ro");
moi>> Plt.title(
...     "pyplot for Modelica");
moi>> Plt.show();
```

A plot window pops up; see Figure 1.

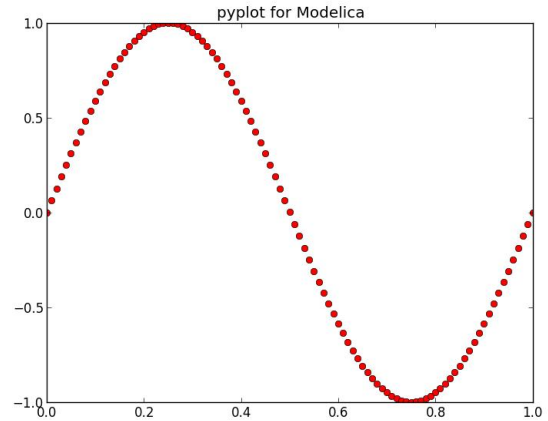


Figure 1: pyplot generated from Modelica

## 2.5 Access Modelica from Python

MoiPy translates Modelica definitions to Python. This means that the translated definitions may also be called from Python directly. When started in interactive mode:

```
python -i moi.py
```

and done some Modelica scripting, e.g.:

```
moi>> import
...     Modelica.SIunits.Conversions.*
moi>> from_degF(70)
```

one may leave MoiPy with `Ctrl-D` (`Ctrl-C` under Windows). A Python command prompt appears. Type:

```
>>> from_degF(70)
```

to directly call the translated Python function. One may switch back to Modelica with:

```
>>> moipy()
```

## 2.6 Advanced Modelica functions

`Modelica.Media` defines more advanced functions. They serve as example for 3D plotting and for debugging in the subsequent section. Take the example:

```
moi>> Modelica.Media.Water.
...     IF97_Utility.h_pT(1e5, 300);
```

In order to evaluate the function for the specific enthalpy, MoiPy needs to load 7 Modelica files and translate 19 functions as well as 7 data records out of 13 packages in MSL 3.2.1. The parsing of the files takes a few seconds. Once loaded and translated, subsequent calls go fluently.

```

moi>> p = linspace(1, 300, 30)
moi>> T = linspace(0, 600, 30);
moi>> (pp, TT) =
... numpy.meshgrid(p, T);
moi>> hh = Modelica.Media.Water.
... IF97_Utilityies.h_pT(
... pp * 1e5,
... TT .+ 273.15);

```

This evaluates the function `h_pT` at  $30 \times 30 = 900$  grid points. See also `examples/mplot3dDemo.moi` resp. call it:

```
moi>> import examples.mplot3dDemo
```

A wireframe plot should pop up; see Figure 2.

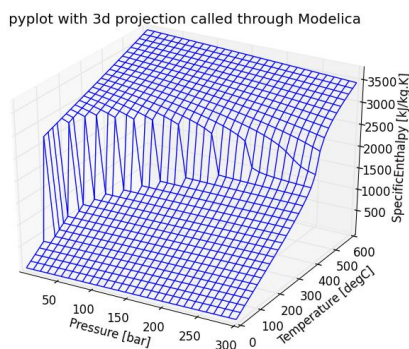


Figure 2: Wireframe plot for `IF97_Utilityies.h_pT`

## 2.7 Debugging of Modelica functions

Python offers the extensible debugger `pdb`. This is exploited by `MoiPy` to transform the debugger outputs to the originating Modelica code. It uses the prompt (`modb`). This gives features like entering the debugger in case of errors, treating break points, stepping through Modelica code, walking up and down on the call stack and inspecting variables.

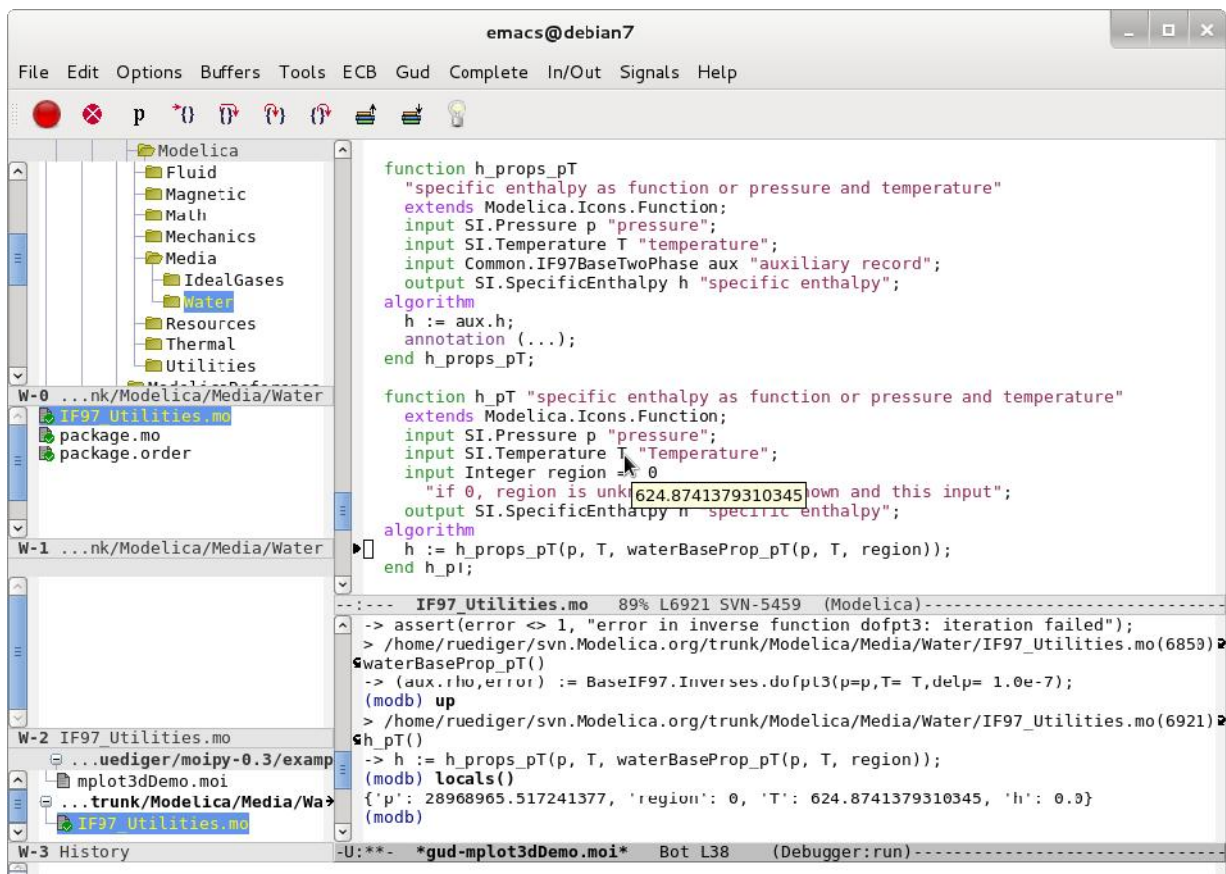
See also below for an example – the debugger shows an error in the `IF97_Utilityies.h_pT` function that was present prior to `r6066` around 290 bar and 350 °C – it has been fixed in `MSL 3.2.1`.

## 2.8 Integration with an IDE

An Integrated Development Environment (IDE) typically integrates several command line tools and can be extended to support new ones. shows Emacs running `modb` as example.

Several Emacs extensions have been loaded, such as Emacs Code Browser for the directory tree, SVN status and revision, besides Modelica mode for syntax highlighting and annotation folding. The Emacs debugger framework parses the output of `modb` and add a graphical user interface. Moreover it opens and shows the respective source file at the right position.

Figure 3: `MoiPy` in the GNU Emacs IDE



### 3 MoiJS – Modelica in JavaScript

Besides its convenient syntax for scientific computing, Modelica is strong in supporting graphical user interfaces. A Modelica model may contain annotations with flowsheet graphics and parameter dialogs, besides documentation – this is important for the specification and exchange of engineering data.

Graphical user interfaces are currently undergoing fundamental changes. Powerful, standardized GUI clients are running on virtually any device, exploiting HTML5 (HTML, CSS and JavaScript). What does this mean for a Modelica client?

Instead of dealing with proprietary server interfaces, events and callbacks, the client could receive a Modelica definition, parse it, build the user dialog, manage user interactions autonomously, and post back user inputs as Modelica definition or modification.

A Modelica parser in JavaScript is needed. Initially developed at Netscape almost 20 years ago, JavaScript grew to a multi-paradigm language covering functional, imperative and object-oriented programming. It gained a lot of momentum since the standardization of HTML5; see [3], the appearance of fast just-in-time compilers, development tools, powerful libraries, such as jQuery [4], and the adoption for server side programming as well, e.g. by Node.js [5].

#### 3.1 Implementation overview

MoiJS implements the Modelica syntax (Modelica 3.3 specification, Appendix B; see [1]) using Jison. Jison provides Flex and Bison (Lex and Yacc) in JavaScript; see [6]. The exemplary rule given in section 2.1 reads:

```

annotation:
  ANNOTATION class_modification
  {
    $$ = new Annotation(track(@$));
    $$ .classModification = $2;
  }
;

```

MoiJS generates a reduced AST as well. A significant difference to MoiPy is that JavaScript objects forming the nodes of the AST are based on prototypes. The prototypes can be extended later on. This means that arbitrary backends can be added without having to touch the original parser code (or requiring the parser to offer a specific plug-in architecture).

#### 3.2 Adding a backend

Assume all executable models of a Modelica library shall be identified, in order to automate testing. Figure 4 shows a MoiJS backend for this.

Figure 4: Exemplary MoiJS backend

```

// load Modelica parser as CommonJS module
var moparser = require("./moparser").parser;

// add method forModifier to each modification and annotation
moparser.Modification.prototype.forModifier =
moparser.Annotation.prototype.forModifier = function (name, callback) {
  (this.classModification || []).forEach(function(argument) {
    if (argument.name == name)
      callback(argument.modification);
  });
}

// add method logStopTime to each class_definition, calling forModifier
moparser.ClassDefinition.prototype.logStopTime = function(within) {
  var definition = this;
  // check for experiment StopTime annotation
  if (definition.annotation) {
    definition.annotation.forModifier("experiment", function(experiment) {
      experiment.forModifier("StopTime", function(modification) {
        console.log(within + "\n  " + definition.ident
          + "(StopTime=" + modification.expression.value + ");");
      });
    });
  }
}
// treat subclasses recursively
(definition.classDefinitionList || []).forEach(function(classDefinition) {
  classDefinition.logStopTime(within + "." + definition.ident);
});
}

```

It may appear confusing how to dive into the syntax tree. MoiJS closely follows the rules and names of the Modelica concrete syntax specification, in order to simplify the understanding. The capitalization is changed to camel style. If for instance the concrete syntax defines a class `modification`, then a class-Modification instance of `ClassModification` appears in the AST. The suffix `List` is used if curly braces find in the Modelica concrete syntax.

The syntax tree may also be explored in the JavaScript Object Notation (JSON) – see below.

The backend (and the parser) can be executed in Node.js that provides, besides a JavaScript runtime, a portable operating system interface through POSIX wrappers. An exemplary Modelica file is processed with:

```
var fs = require("fs");

fs.readFile("Modelica/StateGraph.mo",
  function(err, data) {
    if (err) throw err;
    // parse the file
    var ast =
      moparser.parse(data.toString());
    // call the new backend
    for (i in ast.classDefinitionList)
      ast.classDefinitionList[i]
        .logStopTime(ast.name || "");
  });
```

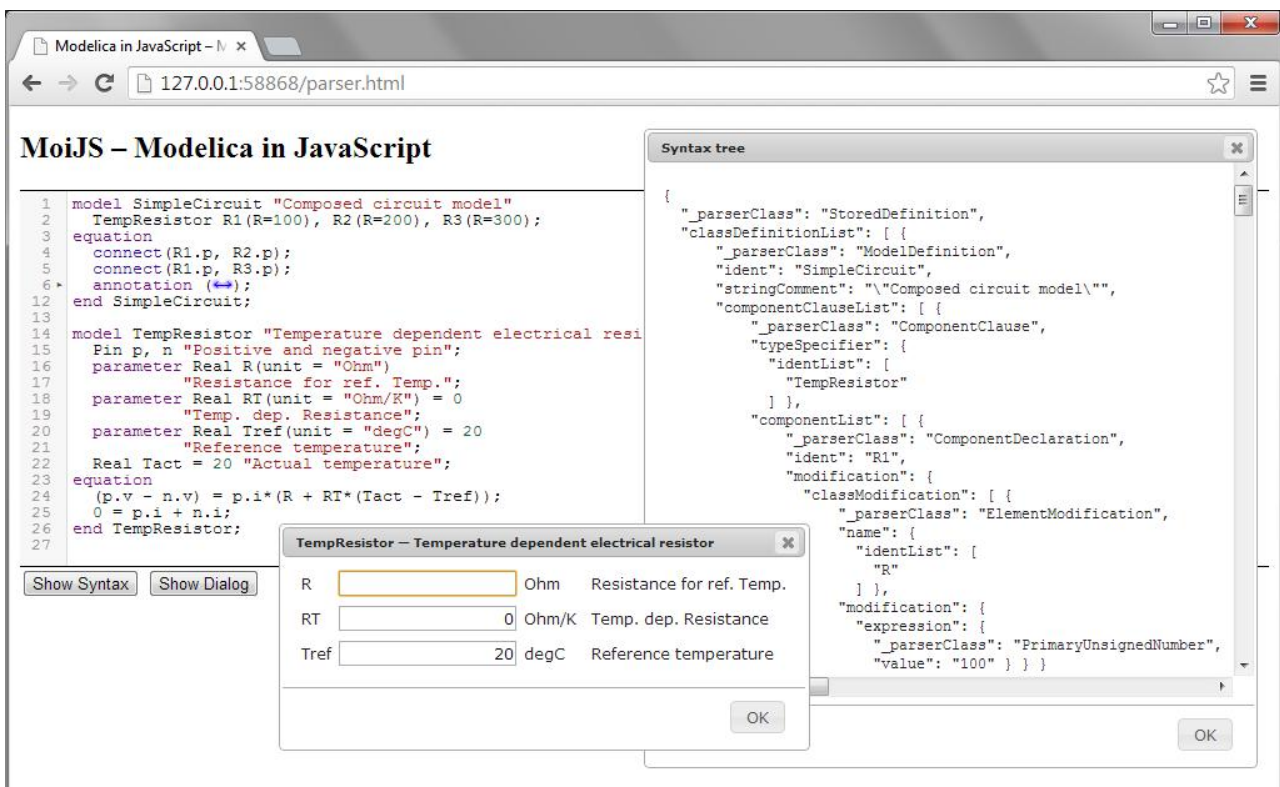
This produces the output:

```
Modelica.StateGraph.Examples.
  FirstExample(StopTime=5);
Modelica.StateGraph.Examples.
  FirstExample_Variant2(StopTime=5);
Modelica.StateGraph.Examples.
  FirstExample_Variant3(StopTime=5);
Modelica.StateGraph.Examples.
  ExecutionPaths(StopTime=15);
Modelica.StateGraph.Examples.
  ShowCompositeStep(StopTime=15);
Modelica.StateGraph.Examples.
  ShowExceptions(StopTime=20);
Modelica.StateGraph.Examples.
  ControlledTanks(StopTime=100);
```

### 3.3 Modelica in a Web browser

Having a Modelica parser in JavaScript, it is a small step to run it in a Web browser, using solely HTML5 standards. Figure 5 shows first results. The MoiJS lexer provides information for code coloring and annotation folding in an HTML textarea. The button “Show Syntax” invokes the MoiJS parser and shows the JSON representation of the resulting syntax tree in a popup window. The button “Show Dialog” invokes a backend to generate a parameter dialog out of the syntax tree. The GUI has been implemented using the jQuery UI library [4].

Figure 5: Modelica in the Google Chrome browser



## 4 Caveats on the Modelica syntax

It is a pleasure to experience that the Modelica concrete syntax can be passed to a general-purpose parser generator, such as PLY or Jison, and something useful comes out. Nevertheless there find some things that might be improved.

### 4.1 Syntax of primary numbers

The Modelica syntax does generally not rely on the use of whitespaces. With one exception: a primary unsigned number may end with a dot and an arithmetic operator might begin with a dot; see also [1], section 10.6.6. The expression

```
2. + [1, 2; 3, 4]
```

is wrong, because the dimensions of the scalar “2.” and the array [1,2;3,4] do not match. The expression

```
2 . + [1, 2; 3, 4]
```

is fine. The additional space makes clear that the dot shall belong to the element-wise addition operator.

The Modelica syntax should be changed to not allow primary numbers ending with a dot. Generally a “2” without dot is fine, in particular because the Modelica division operator “/” is non-truncating, e.g. “1/2” gives 0.5 and not 0. If nevertheless a primary number shall be forced to be a Real, then one can add a 0 behind the dot, e.g. write “2.0”.

### 4.2 Expression syntax

The Modelica expression syntax defines operator precedence and associativity with grammar rules. This leads to very long productions. When a primary number is passed as function argument, for instance, then this primary goes through `factor`, `term`, `arithmetic_expression`, `relation`, `logical_factor`, `logical_term`, `logical_expression`, and `simple_expression`, to finally become an expression.

This is not only hard to read, but also slows down parsers. The expression syntax can alternatively be specified with all operators in one rule, i.e.

```
expr : primary
     | expr or expr
     | expr and expr
     | not expr
     | expr rel_op expr
     | expr add_op expr
     | add_op expr
     | expr mul_op expr
     | expr ("^" | ".^") expr
```

The operator precedence and associativity can be defined in a separate table.

Prec	Operators	Associativity
7	or	left
6	and	left
5	not	right
4	< <= == <> >= >	left
3	+ - .+ .-	left
2	* / .* ./	left
1	^ .^	right

MoiJS, for instance, parses the Modelica Standard Library, version 3.2.1, more than 20% faster with this handy expression syntax.

### 4.3 Syntax of embedded HTML documentation

Working with HTML5 one gets used to documents that contain multiple special-purpose syntaxes, like HTML for content, CSS for styling and JavaScript for behavior – and maybe Modelica for engineering physics. Looking ugly initially, the richer syntax finally helps to faster grasp the different facets of the document. Modern text editors support such documents with mixed modes.

Modelica models may contain embedded HTML documentation. Unfortunately the HTML code needs to be encoded into Modelica strings, meaning that all double quotes used inside the HTML documentation need to be escaped. A general purpose text editor cannot detect and highlight the HTML code.

It should be considered to switch the syntax of embedded HTML documentation from Modelica strings to regular HTML, i.e. allow double quotes up to the ending `</html>` tag. In the simplest case a new Modelica string delimiter could be introduced, like `"""` for multi-line strings in Python. This way the readability improves and the mixed mode support of modern text editors could be exploited.

## 5 Conclusions and Outlook

Today’s Modelica simulation environments offer proprietary client interfaces, basing on Modelica Script, COM or CORBA, besides more. Only XML has been considered as standardized interface format for tool coupling so far. The major drawbacks of XML are its clumsy syntax and that the semantics still needs to be defined.

This paper investigates the use of the Modelica language itself as interface format for client/server architectures and for model exchange, instead of XML. This offers the advantage of having the semantics already standardized. The price to pay is a Modelica parser on the client side. This price turns out to be affordable in modern scripting or Web environments.

Two Modelica parsers have been implemented: MoiPy in Python and MoiJS in JavaScript. Both use the same grammar rules, exploiting parser generators with Lex and Yacc functionality.

It is not the aim of client-side Modelica to compete with simulation environments. The focus is on additional tasks in model-based applications, like scripting, testing, documentation, visualization and graphical user interfaces.

Python is strong in scientific computing. Due to its rich syntax, including e.g. operator overloading, and available packages, such as NumPy, it was straightforward to add a Python backend to the MoiPy parser, resulting in a Modelica interpreter and debugger for algorithmic models.

JavaScript is strong in dynamic user interfaces and in connecting them to servers. This is the main motivation for MoiJS. This paper shows how Modelica code is processed either in a console application or in an HTML5 user interface running the same parser.

The availability of compatible JavaScript implementations by multiple vendors and fast just-in-time compilers being preinstalled on virtually any device make JavaScript attractive for more applications. Examples are HTML5 pages containing verbatim Modelica code that is evaluated on the client side, e.g. in a Web browser or in a modern mobile device.

MoiJS is extensible with new backends, exploiting the prototype based inheritance of JavaScript. MoiJS is available under the MIT license at <http://omuses.github.io/moijs>.

## References

- [1] Modelica – A Unified Language for Systems Modeling, Language Specification, version 3.3, May 9, 2012. <https://www.modelica.org/documents/ModelicaSpec33.pdf>
- [2] David Beazley: PLY (Python Lex Yacc), version 3.4, 2011. <http://www.dabeaz.com/ply/>
- [3] Steve Jobs: Thoughts on Flash, April 2010. <http://www.apple.com/hotnews/thoughts-on-flash/>
- [4] jQuery – write less, do more; and jQuery UI. <http://jquery.com>, <http://jqueryui.com>
- [5] Node.js – a platform for easily building fast, scalable network applications. <http://nodejs.org>
- [6] Zachary Carter: Jison – your friendly JavaScript parser generator, version 0.4.13, 2013. <http://zaach.github.io/jison/>

## Acknowledgements

This work was supported in parts by the Federal Ministry of Education and Research (BMBF) within the ITEA2 project MODRIO (Model Driven Physical Systems Operation) – BMBF funding code: 01IS12022A.



# Dynamic modelling of a Condenser with the ThermoSysPro Library

Baligh El Hefni      Daniel Bouskela

EDF R&D

6 quai Watier, 78401 Chatou Cedex, France  
baligh.el-hefni@edf.fr      daniel.bouskela@edf.fr

## Abstract

The condenser is an important device for the operation of power plants in particular for pressurized water reactors. Undesirable transients may lead to the automatic shutdown of the power plant.

To simulate the complex dynamic physical behaviour of the condenser, a dynamic model has been developed using Modelica. The component model is meant to be used for power plant modeling and simulation with the ThermoSysPro library developed by EDF and released under open source license.

The transient most unfavorable to keeping the integrity of the condenser vacuum is the loss of the cold source pumps followed by a turbine trip and a missed islanding (house load operation).

The objective is to study the evolution of the pressure of the condenser for this kind of transients. During the transient, the condenser pressure should be always less than  $5 \times 10^4$  Pa.

The present paper describes in detail the condenser model: hypothesis, governing equations, correlations and the test-case (structure of the test model, the parameterization data and the results of simulation).

**Keywords:** *Modelica; thermal-hydraulics; heat exchange; condenser; dynamic modeling; inverse problems; ThermoSysPro*

## 1. Introduction

Modeling and simulation activities play a key role in the design phase and performance optimization of complex energy processes. It is also expected that they will play a significant role in the future for power plant maintenance and operation.

The potential of Modelica as a mean to efficiently describe thermodynamic models has been recognized for quite a while [1, 2], and has led to the initiative of developing an EDF library for power plant modeling within the ITEA 2 EUROSYSLIB project.

This library, called ThermoSysPro, aims at providing the most frequently used models of components for the 0D-1D static and dynamic modeling of thermodynamic systems, mainly for power plants, but also for other types of energy systems such as industrial processes, energy conversion systems, buildings etc. It involves disciplines such as thermal-hydraulics, combustion, neutronics and solar radiation (see instance [1 to 8]).

The ambition of the library is to cover all the phases of the plant lifecycle, from basic design to plant operation. This includes for instance system design, verification and validation of the instrumentation and control system, system diagnostics and plant monitoring. To that end, the library will be linked in the future to systems engineering via the modeling of systems properties (or requirements), and to the measurements made on the real process via state estimation techniques.

The present paper focuses on the dynamic modeling of the condenser: hypothesis, governing equations, correlations and the test-case (structure of the model, the parameterization data and the results of simulation).

## 2. Model of the condenser

### 2.1. General presentation of the condenser

The condenser is a **two-phase** shell-and-tube heat exchanger. The feedwater flows inside the tube bundle, while the steam and condensate flows outside of those tubes located inside the cavity. In the condenser, there are two zones: the desuperheating zone and the condensation zone.

The condensate of water heaters located upstream is injected into the condenser. During the injection, part of the condensate may vaporize due to the pressure drop. This phenomenon is known as flash.

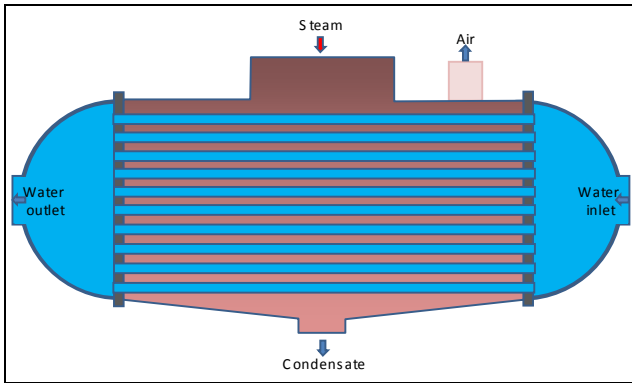


Figure 1: fluid flow inside an condenser

## 2.2. Description of the condenser model

The **DynamicCondenser** model represents the dynamics of the thermo-hydraulic phenomena of the hot fluid inside the cavity and of the cooling fluid which flows through the tube bundle. In particular, the model features the thermal exchanges between the fluid in the cavity and the cooling fluid flowing through the tube bundle.

The condenser is considered as a vertical or horizontal cylindrical cavity (as schematized in Figure 1), containing a tube bundle with the feedwater inlet and outlet located on the end.

## 2.3. Components of the condenser

The model is divided into sub-models of three different types which are connected together to make the full model (see Figure 2):

- One **DynamicOnePhaseFlowPipe** model,
- One **HeatExchangerWall** model,
- One **TwoPhaseCavityOnePipe** model.

By reassembling the sub-models, any other configuration of the condenser can be modelled.

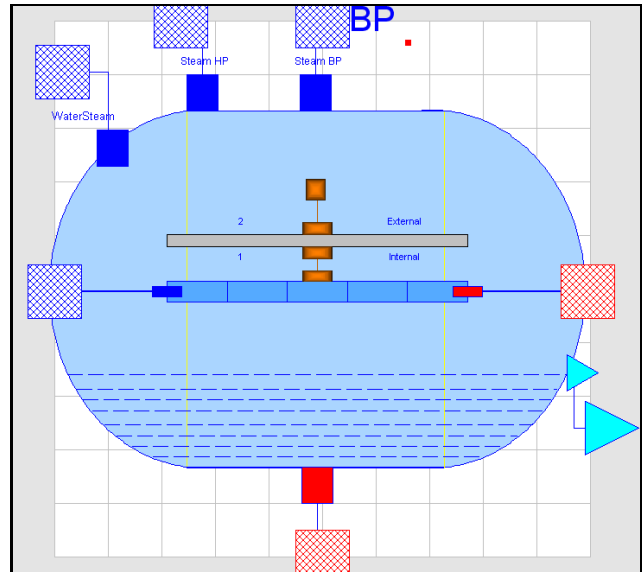


Figure 2: icon representation of the component model

The description of each sub-model is given in the following section. Each sub-model in the model can be recognized by looking at its icon (see Figures 3, 4 and 6).

## 3. Physics of the condenser

### 3.1. **DynamicOnePhaseFlowPipe** model

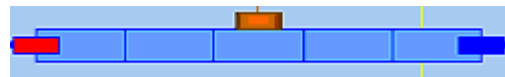


Figure 3: pipe model icon

The model of the fluid flow in a cylindrical conduit is based on the dynamic mass, energy, and momentum balance equations, which are originally given as 1-D partial differential equations. The original distributed-parameter model is first discretised by using the finite-volume method. The model is formulated in order to correctly handle possible flow reversal conditions.

### Assumptions

- Homogeneous fluid in each mesh cell (same velocity for the liquid and steam phases);
- 1-D modelling (using the finite-volume method);
- The accumulation is considered in each mesh cell;
- The inertia of the fluid is taken into account;
- The phenomenon of longitudinal heat conduction in the metal wall and in the fluid is neglected;
- The thermo-physical properties are calculated on the basis of the average pressure and enthalpy in each mesh cell.

### Mass balance equation

The mass balance equation in each cell is given by:

$$A \cdot \frac{d\rho_i}{dt} \cdot \Delta x = \dot{m}_{i-1i} - \dot{m}_{ii+1}$$

Taking the pressure and the specific enthalpy as state variables yields:

$$A \cdot \left[ \left( \frac{\partial \rho_i}{\partial P_i} \right)_h \cdot \frac{dP_i}{dt} + \left( \frac{\partial \rho_i}{\partial h_i} \right)_P \cdot \frac{dh_i}{dt} \right] \cdot \Delta x = \dot{m}_{i-1i} - \dot{m}_{ii+1}$$

### Energy balance equation

The energy balance equation in each cell is given by:

$$A \cdot \frac{d(\rho_i \cdot u_i)}{dt} \cdot \Delta x = \dot{m}_{i-1i} \cdot h_{i-1i} - \dot{m}_{ii+1} \cdot h_{ii+1} + \Delta W_i$$

with the specific internal energy given by:

$$u_i = h_i - \frac{P_i}{\rho_i}$$

Taking the pressure and the specific enthalpy as state variables yields:

$$A \cdot \left( \left( h_i \cdot \frac{\partial \rho_i}{\partial P_i} - 1 \right) \cdot \frac{dP_i}{dt} + \left( h_i \cdot \frac{\partial \rho_i}{\partial h_i} + \rho_i \right) \cdot \frac{dh_i}{dt} \right) \cdot \Delta x = \dot{m}_{i-1i} \cdot h_{i-1i} - \dot{m}_{ii+1} \cdot h_{ii+1} + \Delta W_i$$

$h_{i,i+1}$  is the specific enthalpy of the mass flow  $\dot{m}_{i,i+1}$  crossing the boundary between the cells  $i$  and  $i+1$ .  $h_{i,i+1}$  is related to the state variables  $h_i$  and  $h_{i+1}$  by:

$$h_{i,i+1} = \hat{s}(P_e) \cdot h_i + \hat{s}(-P_e) \cdot h_{i+1}$$

where  $P_e$  is the Peclet number and

$$\hat{s}(x) = \frac{1}{1 + e^{-\frac{x}{2}}} \quad (\text{see e.g. [10]}).$$

When neglecting diffusion, the Peclet number is infinite, and

$$h_{i,i+1} = s(\dot{m}_{i,i+1}) \cdot h_i + s(-\dot{m}_{i,i+1}) \cdot h_{i+1}$$

Where  $s$  is the step function:

$$s(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

This simplification is known as the upwind scheme.

### Momentum balance equation

The momentum balance equation in each cell is given by:

$$1/A \cdot \frac{d\dot{m}_{i,i+1}}{dt} \cdot \Delta x =$$

$$P_i - P_{i+1} - (\Delta P)_{i,i+1}^a - (\Delta P)_{i,i+1}^f - (\Delta P)_{i,i+1}^g$$

with respectively the acceleration, friction and gravity pressure losses given by:

$$(\Delta P)_{i,i+1}^a = \frac{1}{A^2} \cdot \dot{m}_{i,i+1} \cdot |\dot{m}_{i,i+1}| \cdot \left( \frac{1}{\rho_{i+1}} - \frac{1}{\rho_i} \right)$$

$$(\Delta P)_{i,i+1}^f = \zeta \cdot \frac{\Lambda_i \cdot \Delta x_h}{2 \cdot D \cdot A^2 \cdot \rho_i} \cdot \dot{m}_{i,i+1} \cdot |\dot{m}_{i,i+1}|$$

$$(\Delta P)_{i,i+1}^g = \rho_{i,i+1} \cdot g \cdot (z_{i+1} - z_i)$$

By default, the flow is considered turbulent (Reynolds number  $Re > 2300$ ).

The **Colebrook** correlation is used to compute  $\Lambda_i$ .

### Convective heat transfer within the tubes

The heat exchanged between the fluid and the wall is:

$$\Delta W(i) = h_c(i) \cdot \Delta S_2 \cdot (T_{w2}(i) - T(i))$$

### Convection heat transfer coefficient

The convection heat transfer coefficient  $h_c$  between the fluid and the wall is computed using the **Dittus-Boelter** correlation.

### 3.2. HeatExchangerWall model

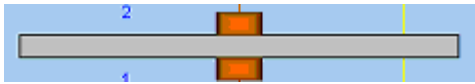


Figure 4: wall model icon

The wall model describes the conductive heat flow through the wall of the tube bundle. The flow is positive when entering the tubes (going from side 2 to side 1 of the wall).

$$\Delta W_1(i) = \frac{2 \cdot \pi \cdot \lambda \cdot \Delta x(i) \cdot ntubes \cdot (T_w(i) - T_{w1}(i))}{\ln((e + D)/D)}$$

$$\Delta W_2(i) = \frac{2 \cdot \pi \cdot \lambda \cdot \Delta x(i) \cdot ntubes \cdot (T_{w2}(i) - T_w(i))}{\ln((2 \cdot e + D)/(e + D))}$$

$$\Delta M_w \cdot c_{pw} \cdot \frac{dT_w}{dt} = \Delta W_2(i) - \Delta W_1(i)$$

### 3.3. TwoPhaseCavityOnePipe model

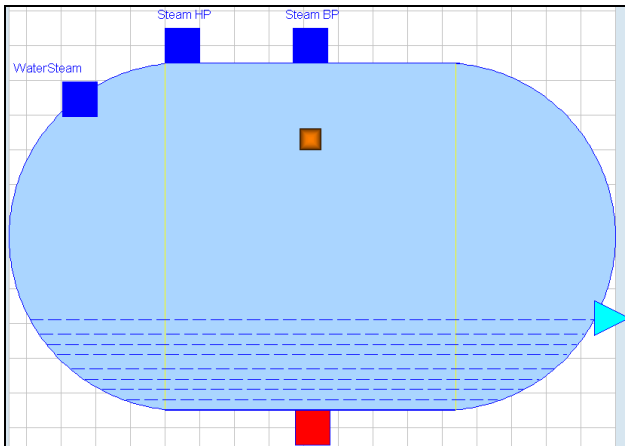


Figure 5: two-phase cavity model icon

The cavity is modelled as a non-adiabatic two-phase volume, with vertical or horizontal cylindrical geometry. The physical model is based on a non-equilibrium, two-phase formulation of the fluid balance equations with a control volume approach. The two phases are supposed to be isobaric and will be referred to as liquid zone and steam zone, respectively.

The model features the condensation flow of the steam phase into the liquid phase.

In most operating conditions, the liquid and steam phases in cavity are not necessarily in thermal equilibrium. The reasons are:

- The vapour may enter the cavity in a superheated state (the vapour temperature is then higher than the saturation temperature).
- The liquid may be subcooled by the incoming drain.

#### Assumptions

- Accumulation of mass and energy is considered. Heat exchange between the liquid and steam phases is considered.
- Heat exchange between the liquid or steam phases and the wall is considered.
- Heat exchange between the condenser and the external medium (ambient) is considered.
- Pressure losses are not taken into account in the cavity.
- The liquid and steam phases are not necessarily in thermal equilibrium.
- The liquid and steam phases are assumed to be permanently in pressure equilibrium.

#### State variables

The state variables of the system are:

- the mean pressure in the cavity,
- the specific enthalpy of the liquid phase,
- the specific enthalpy of the steam phase,
- the temperature of the wall,
- the volume of the liquid phase.

The volume of the steam phase is bound to the volume of the liquid phase by the following equation:

$$V_l + V_v = V$$

#### Mass balance equation in each phase

$$\frac{d(\rho_l \cdot V_l)}{dt} = -\dot{m}_l^o + (1 - x_{mv}) \cdot \dot{m}_{drain}^e + \dot{m}_{cond}$$

$$\frac{d(\rho_v \cdot V_v)}{dt} = \dot{m}_v^e + x_{mv} \cdot \dot{m}_{drain}^e - \dot{m}_{cond}$$

where  $\dot{m}_v^e$  is the mass flow of incoming vapor,  $\dot{m}_{drain}^e$  is the mass flow of the incoming condensate of the some water heaters,  $\dot{m}_l^o$  is the mass flow of outgoing condensate and  $\dot{m}_{cond}$  is the condensation flow inside the cavity.

**Condensation mass flow rate inside the cavity**

The evolution of the pressure in the condenser depends strongly upon the mass flow rate of condensation.

The classical equation used to calculate the condensation mass flow rate given in [8] is not correct to simulate the scenario "loss of the cold source pumps followed by a turbine trip and a missed islanding". The following new equation gives correct results:

$$\dot{m}_{cond\_1} = \frac{W_{1t} + W_{2t} + W_{vl} + W_{vp}}{h_v^{sat} - h_l^{sat}}$$

$$\dot{m}_{cond\_2} = (1 - x_e^v) \cdot \dot{m}_v^e$$

$$\dot{m}_{cond} = \dot{m}_{cond\_1} + \dot{m}_{cond\_2}$$

where  $\dot{m}_{cond\_1}$  is the mass flow rate of the steam condensed in the cavity and  $\dot{m}_{cond\_2}$  is the mass flow rate of the water contained in the wet steam at the cavity inlet.

**Energy balance equation in each phase**

The general form of the energy balance equation is given by:

$$\frac{d(\rho \cdot V \cdot u)}{dt} = \sum_e \dot{m}_e \cdot h_e + \sum_o \dot{m}_o \cdot h_o + \sum W$$

Taking the pressure and the specific enthalpy as state variables yields:

$$V_l \cdot \left[ \left( \frac{P}{\rho_l} \cdot \left( \frac{\partial \rho_l}{\partial P} \right)_h - 1 \right) \cdot \frac{dP}{dt} + \left( \frac{P}{\rho_l} \cdot \left( \frac{\partial \rho_l}{\partial h_l} \right)_p + \rho_l \right) \cdot \frac{dh_l}{dt} \right] =$$

$$-\dot{m}_l^o \cdot \left( h_l^o - \left( h_l - \frac{P}{\rho_l} \right) \right) + \dot{m}_{cond} \cdot \left( h_l^{sat} - \left( h_l - \frac{P}{\rho_l} \right) \right)$$

$$+ (1 - x_{mv}) \cdot \dot{m}_{drain}^e \cdot \left( h_{drain\_l}^e - \left( h_l - \frac{P}{\rho_l} \right) \right) + W_{vl} - W_{lw}$$

$$V_v \cdot \left[ \left( \frac{P}{\rho_v} \cdot \left( \frac{\partial \rho_v}{\partial P} \right)_h - 1 \right) \cdot \frac{dP}{dt} + \left( \frac{P}{\rho_v} \cdot \left( \frac{\partial \rho_v}{\partial h_v} \right)_p + \rho_v \right) \cdot \frac{dh_v}{dt} \right] =$$

$$\dot{m}_v^e \cdot \left( h_v^e - \left( h_v - \frac{P}{\rho_v} \right) \right) - \dot{m}_{cond} \cdot \left( h_l^{sat} - \left( h_v - \frac{P}{\rho_v} \right) \right)$$

$$+ x_{mv} \cdot \dot{m}_{drain}^e \cdot \left( h_{drain\_v}^e - \left( h_v - \frac{P}{\rho_v} \right) \right) - W_{vl} - W_{vw} - W_{1t}$$

**Energy accumulation at the wall**

$$M_w \cdot c_{pw} \cdot \frac{dT_w}{dt} = W_{lw} + W_{vw} - W_{wa}$$

**Heat exchange between the liquid and steam phases**

$$W_{vl} = K_{vl} \cdot A_{vl} \cdot (T_v - T_l)$$

**Heat exchange between the liquid or steam phases and the wall**

$$W_{lw} = K_{lw} \cdot A_{lw} \cdot (T_l - T_w)$$

$$W_{vw} = K_{vw} \cdot A_{vw} \cdot (T_v - T_w)$$

**Heat exchange between the condenser and the external medium**

$$W_{wa} = K_{wa} \cdot A_{wa} \cdot (T_w - T_a)$$

**Heat exchange between the liquid and the tube bundle 'Pipes'**

$$\Delta W_1(i) = \beta \cdot h_{conv}(i) \cdot \Delta S_{ext} \cdot (T_v - T_{w1}(i)) + W_{2t} / N$$

$$W_{1t} = \sum \Delta W_1(i)$$

**Total power exchanged for deheating, for  $h_v^e > h_v^{sat}$**

$$W_{2t} = \dot{m}_v^e \cdot (h_v^e - h_v^{sat})$$

**Heat transfer convection coefficients**

The **Nusselt** correlation is used to calculate the heat transfer coefficients  $h_{conv}$  between the steam and the outside wall of the tube bundle, in the condensation zone.

The heat transfer coefficient between the liquid and the wall ( $K_{lw}$ ), the heat transfer coefficient between the steam and the wall ( $K_{vw}$ ) and the heat transfer coefficient between the steam and the liquid ( $K_{vl}$ ) were calculated for flat wall.

## 4. Test-case of the condenser

### 4.1. Modelica model of the condenser

To simulate the complex dynamic physical behaviour in normal and dysfunctional conditions of the condenser model, a test model called “TestDynamicCondenser” has been developed by assembling the necessary components from the **ThermoSysPro** library (cf. Figure 6). The test model includes the level control system.

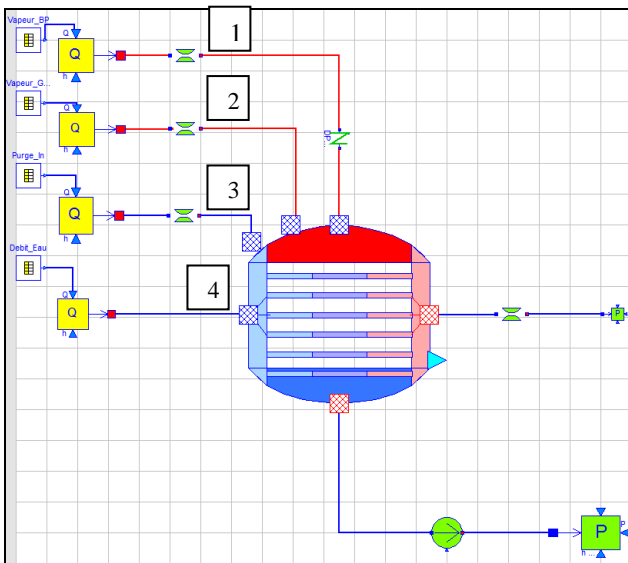


Figure 6: model of the condenser “TestDynamicCondenser”

### 4.2. Data implemented in the model

All geometrical data were provided to the model: tubes and exchangers lengths, diameters, volumes, corrective terms for the heat exchange coefficients, corrective terms for the pressure losses, etc. The plant characteristics are given in Figure 9 (cf. Appendix).

### 4.3. Calibration of the model

The calibration step consists in setting (blocking) the maximum number of thermodynamic variables to known measurement values. This method ensures that all needed performance parameters, size characteristics and output data can be computed.

For this model:

- setting (blocking) the cavity pressure to known measurement value and the value of corrective term of heat exchange coefficient between the cavity and pipes can be computed by model inversion,
- setting (blocking) the mass flow rate in the pump and the characteristics of the pump can be computed by model inversion (a polynomial coefficient)

### 4.4. Simulation scenario

In order to challenge the dynamic simulation capabilities of the model, a high amplitude transient "loss of the cold source pumps followed by a turbine trip and a missed islanding" is applied as a simulation scenario to the model. The purpose is to study the evolution of the pressure of the condenser for this type of transient. For this class of transients, the condenser pressure should always be less than  $5e4$  Pa (maximum pressure for the availability of the condenser).

This transient is used to check and validate the physics taken into account in the model and the numerical robustness of the model as it runs the condenser model into very different operating regimes. This allows to assess the validity and applicability range of the model equations, and the numerical robustness of the Modelica implementation when using Dymola.

### 4.5. Boundary conditions of the model

The boundary conditions of the model (scenario profiles) are presented in Figure 7.

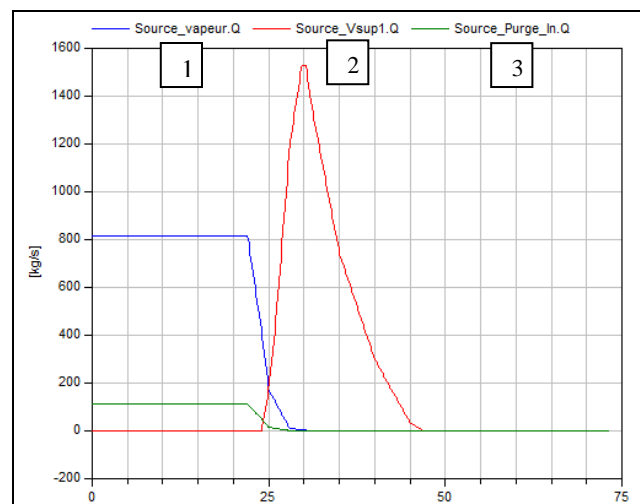


Figure 7a: input mass flow rate of the steam

Others data of the model:

Enthalpy of the LP steam “1”	2416.6e3	J/kg
Enthalpy of the HP steam “2”	2759.6e3	J/kg

Enthalpy of the drain “3”	228.21e3	J/kg
Enthalpy of the feed water “4”	141.75e3	J/kg

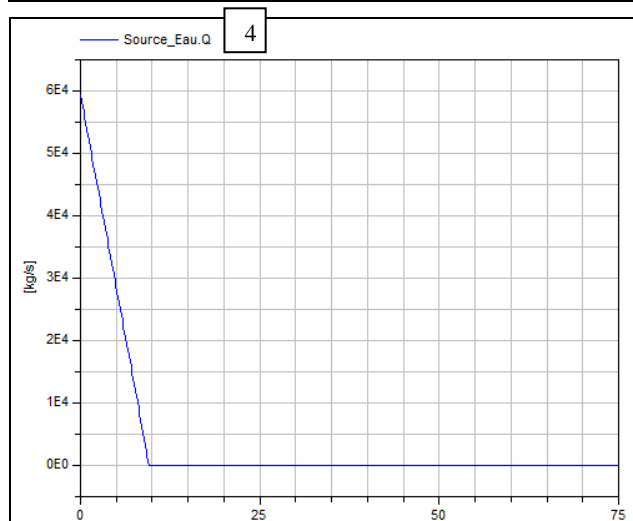


Figure 7b: input mass flow rate of the feedwater

#### 4.6. Results of dynamic simulations

In order to cover the whole transient, the simulation time has been set at 75 seconds.

Simulation runs were done using Dymola 6.1 and Dymola 2012, with Dassl solver and with a tolerance = 0.0001. The simulation of the scenarios were mostly successful, with only one iteration variable to be fed manually.

The following phenomena are simulated:

- local condensation,
- swell and shrink effect in cavity,
- cavity water levels.

The model is able to compute:

- the mass flow rate of the condensate,
- the thermal power of the condenser and tubes,
- the distribution of pressure, temperature and specific enthalpy inside the tube bundle,
- the cavity water level and cavity pressure.

The results of the simulation runs are given in Figure 8. Figures 8a and 8b show the results obtained with Dymola. The simulation result demonstrates that the maximum pressure in the condenser is less than 5e4 Pa as required.

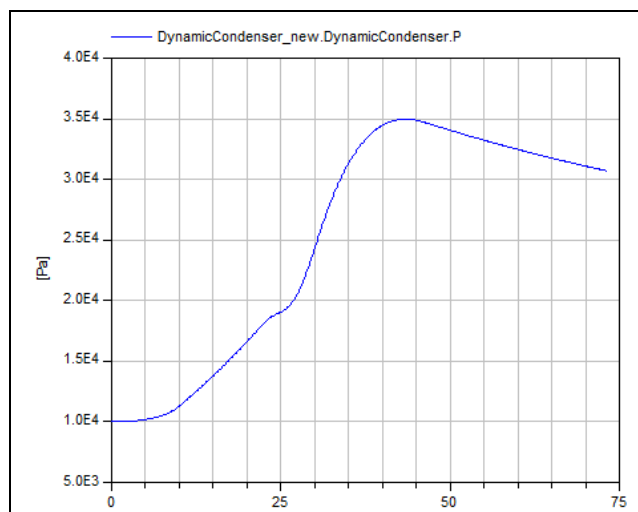


Figure 8a: evolution of the pressure inside the condenser

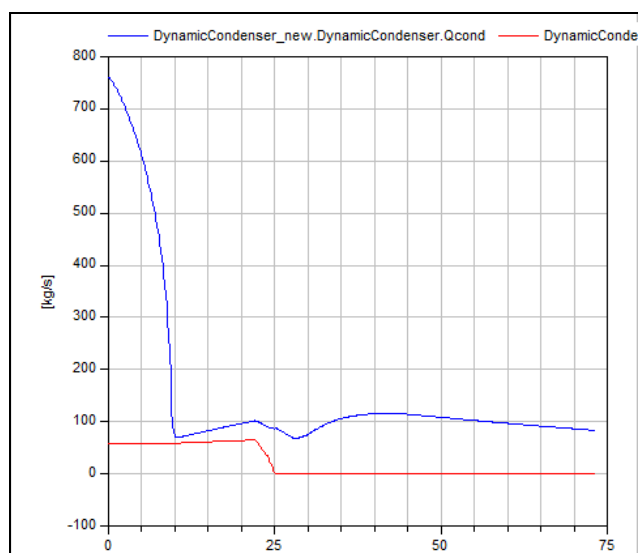


Figure 8b: evolution of the mass flow rate of the condensate ( $\dot{m}_{cond\_1}$ ,  $\dot{m}_{cond\_2}$ )

## 5. Conclusion

A new open source Modelica library called ‘ThermoSysPro’ has been developed within the framework of the ITEA 2 EUROSYS LIB project. This library has been mainly designed for the static and dynamic modeling of power plants, but can also be used for other energy systems such as industrial processes, buildings, etc. It is intended to be easily understood and extendable by the models developer.

A new dynamic model of a condenser has been developed using existing elements of ThermoSysPro.

To validate the model, the following difficult transient was simulated: loss of cold source pumps followed by a turbine trip and a missed islanding.

The simulation result shows that the maximum pressure in the condenser is less than 5e4 Pa (maximum pressure for the availability of the condenser) as required by the operation rules. The model enables to show that the evolution of the pressure inside the condenser strongly depends on the value of the condensation mass flow rate, the heat exchange between the vapor and the liquid at the bottom of the condenser and the thermal loss to the ambient.

## Nomenclature

### Symbols

$\dot{m}$	Mass flow
$N$	Number of segments for cooling pipes
$\rho$	Fluid density
$h$	Fluid specific enthalpy
$u$	Fluid specific internal energy
$P$	Fluid pressure
$T$	Fluid temperature
$c_p$	Fluid specific heat capacity
$V$	Volume
$t$	Time
$W$	Power
$x_v$	Vapor mass fraction in vapor phase
$x_l$	Vapor mass fraction in liquid phase
$x_{mv}$	Vapor mass fraction in input drain
$z_i$	Pipe inlet altitude
$\Lambda$	Friction coefficient
$\zeta$	Friction corrective coefficient
$\beta$	Corrective term for heat exchange coefficient
$\Delta x$	Tube segment length
$\Delta S$	Heat surface exchange of tube segment
$D$	Tube diameter
$A$	Tube cross section or heat exchange surface
$e$	Wall thickness
$\lambda$	Conduction coefficient
$K$	Heat exchange coefficient
$M$	Mass
$h_c$	Convective coefficient
$h_{conv}$	Convective coefficient of heat transfer by condensation between the vapor and

	<i>the tube bundle</i>
$ntubes$	Number of tubes in the bundle

### Indices

$X_i$ or $X(i)$	Quantity in volume $i$
$X_{i:i+1}$	Flow of quantity between volume $i$ and volume $i+1$
$X_e$ or $X^e$	Quantity at inlet
$X_o$ or $X^o$	Quantity at outlet
$X_l$	Quantity relative to liquid
$X_v$	Quantity relative to vapor
$X_w$	Quantity relative to cavity wall
$X_{w1}$	Quantity relative to pipes wall
$X_{ext}$	Quantity relative to external side of wall
$X_a$	Quantity relative to ambient
$X^{sat}$	Quantity relative to saturated phase
$X_{cond}$	Quantity relative to condensation
$X_{drain}$	Quantity relative to drain
$X_{drain\_l}$	Quantity relative to drain, for liquid
$X_{drain\_v}$	Quantity relative to drain, for vapor

## References

- [1] Bouskela D., Chip V., El Hefni B., Favennec J.M., Midou M. and Ninet J. 'New method to assess tube support plate clogging phenomena in steam generators of nuclear power plants', *Mathematical and Computer Modelling of Dynamical Systems*, 16: 3, 257-267, 2010.
- [2] El Hefni B., Bouskela D., 'Modelling of a water/steam cycle of the combined cycle power plant "Rio Bravo 2" with Modelica', *Modelica 2006 conference proceedings*.
- [3] David F., Souyri A., Marchais G., 'Modelling Steam Generators for Sodium Fast Reactors with Modelica', *Modelica 2009 conference proceedings*
- [4] El Hefni B., Péchiné B., 'Model driven optimization of biomass CHP plant design', *Mathmod conference 2009*, Vienna, Austria.
- [5] El Hefni B., Bouskela D., 'Dynamic modelling of a combined cycle power



- plant with ThermoSysPro', Modelica 2011 conference proceedings
- [6] Souyri A., Bouskela D., 'Pressurized Water Reactor Modelling with Modelica', Modelica 2006 conference proceedings.
- [7] El Hefni B. 'Dynamic modeling of concentrated solar power plants with the ThermoSysPro Library' (Parabolic Trough collectors, Fresnel reflector and Solar-Hybrid, SolarPaces 2013, Elsevier's Energy Procedia.
- [8] Baligh El Hefni, , Daniel Bouskela, Guillaume Gentilini 'Dynamic modelling of a water heater with the ThermoSysPro Library' Modelica2012
- [9] Collier J.G., and Thome J.R., 'Convective Boiling and Condensation', Mc Graw-Hill Book Company (UK) limited, 1972 Clarendon Press, Oxford, 1996.
- [10] Patankar S.V., 'Numerical Heat Transfer and Fluid Flow', Hemisphere Publishing Corporation, Taylor & Francis, 1980.

## Appendix

**Component**

Name:

Comment:

**Model**

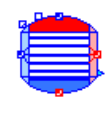
Path: ThermoSysPro.WaterSteam.HeatExchangers.DynamicCondenser

Comment: Dynamic Cavity

**Parameters**

Vf0	0.0463379		Fraction of initial liquid volume in the Cavity ( $0 < Vf0 < 1$ )
P0c	1e4	Pa	Initial pressure in the Cavity
Rv	12.89873616	m	Radius of the Cavity cross-sectional area
Lv	10.7903346	m	Cavity length
L2	15.4	m	Pipes length
Lc	2.5	m	support plate spacing in cooling zone(Chicanes)
Dc	0.018	m	Internal diameter of the cooling pipes
ec	0.5e-3	m	Thickness of the cooling pipes
Ns	5		Number of segments for pipes
ntubest	92916		Number of total pipes in Cavity
ntubesV	220		Numbers of pipes in a vertical plan in Cavity
cp	573	J/(kg.K)	Specific heat capacity of the metal of the cooling pipes
rho	4579	kg/m3	Density of the metal of the cooling pipes
lambda	21.4	W/(m.K)	Wall thermal conductivity of the cooling pipes
DynamicCondenser	=50e3, z(fixed=false, start=0.5), Kpa=0.01, Qcond(start=1000))		
pipe_3	), fixed=false)), advection=false, dynamic_mass_balance=false)		
Ce2	2[h_vol(fixed=false, start=172390), h(fixed=false, start=172390))		

Icon



**Figure 9: data of the model**

# Wavelet Library for Modelica

Jianbo Gao\*, Yang Ji\*\*, Johann Bals\*\*, Ralph Kennel\*

\*Technische Universitaet Muenchen, Arcisstr. 21, 80333 Munich, Germany  
michael.gao@tum.de, ralph.kennel@tum.de

\*\* German Aerospace Center, Muenchner Str. 20, 82234 Wessling, Germany  
yang.ji@dlr.de, johann.bals@dlr.de

## Abstract

Wavelet analysis is being widely used in different fields for signal processing to increase efficiency and flexibility. A wavelet library has been a standard component in many simulation programs. However, wavelet analysis has not yet been included in Modelica as a standard component. To fill this blank, a comprehensive wavelet library has been developed for Modelica. This library includes fifteen commonly used wavelet families. It can carry out continuous transform, forward and inverse discrete transforms, and multi-level decomposition and reconstruction in one-dimensional space. In addition, special application tools for multi-resolution analysis and wavelet denoising are provided. Moreover, some examples are given to provide the users a quick start point to build up their own algorithms. This library was programmed and tested according to the Modelica language specification 3.2 under the Dymola platform version 2013. The test results prove the functionalities of the library.

*Keywords: Modelica; Dymola; wavelet; Modelica library*

## 1 Introduction

Although the wavelet transform has only been established for several decades [1][2], it has already been applied in many fields because of its superior properties in signal processing. Besides practical applications, the wavelet transform is also a versatile tool for different simulation problems [3][4][5][6]. So far the wavelet transform has become a standard toolbox in many free and commercial simulation programs, such as Simulink™, Mathematica™, Maple™, Vis-Sim™ and many others.

Modelica, as a comprehensive multi-physical simulation tool, has not yet included a library for carrying out wavelet transform. However, there exists demand for using wavelet transform within Modelica. In

2005, Bunte and his colleagues analyzed the simulation data of vehicle steering dynamics generated in Modelica with the wavelet transform [7]. Because of the lack of wavelet transform in Modelica, the analysis had to be done with other software. In the paper of Ji [8] in 2010 about a Modelica signal analysis tool towards the design of more electric aircrafts the authors expressed the wish to do the analysis using wavelet transform directly within Modelica.

The first application for the failure analysis in the power system using the prototype of the Modelica wavelet library was already introduced in 2012 [9]. After the successful development, the final Modelica wavelet library is introduced in this article. It includes the following features.

- Fifteen wavelet families;
- Continuous transform;
- Discrete forward and inverse transforms;
- Multi-level decomposition and reconstruction;
- Multi-resolution analysis;
- Denoising and data compression;
- User friendly graphic user interface (GUI);
- Importing simulation data from the hard disk;
- Fully open source under Modelica License 2.

In the first release version, the library has the following limitations.

- Only one-dimensional transformations are implemented.
- Only post-processing is possible. The algorithms cannot execute in real time applications.
- The data to be analyzed must be have equidistant time grids.
- The library is developed and tested under Dymola 2013 demo version.

## 2 Wavelet transform

The basic knowledge about wavelet theory is required in order to use the wavelet library. In this section, a very brief description about the wavelet the-

ory will be given [1][2]. The theoretical part of [9] is roughly repeated here but with improvements for better understanding of common readers.

## 2.1 Definition

The wavelet transform can be considered as a further development of the Fourier transform, or more precisely, of the short time Fourier transform (STFT) [10]. Using the STFT, people try to localize the signal changing by selecting suitable time windows. This transformation, however, has its inherent drawbacks. The most significant one is its limit in time-frequency resolution due to the uncertainty principle. The wavelet transform overcomes this problem. This transformation is defined as [1]:

$$[W_{\psi}f](a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} \overline{\psi\left(\frac{t-b}{a}\right)} f(t) dt. \quad (1)$$

This equation defines the wavelet transform of the function  $f(t)$  using wavelet function  $\psi$  at scale  $a$  and position  $b$ . The bar above function  $\psi$  stands for conjugation. For given  $a$  and  $b$ , the transformation result is a single real number, named wavelet coefficient. The wavelet function must satisfy some conditions to ensure that it is an orthonormal function:

- (1) Zero mean:  $\int_{\mathbb{R}} \psi(t) dt = 0$ ;
- (2) Unit square norm:  $\int_{\mathbb{R}} |\psi(t)|^2 dt = 1$ ;
- (3) Compact support:  $\psi(t) = 0$ , for  $|t| > C$ , where  $C$  is a positive real number.
- (4) Dyadic orthogonality:  $\langle \psi_{jk}, \psi_{lm} \rangle = \delta_{jl} \delta_{km}$ ,  
for  $\psi_{jk} = 2^{\frac{j}{2}} \psi(2^j x - k)$ . Here  $\langle \cdot, \cdot \rangle$  means inner product of two functions.

The wavelet function only represents the signal components that have zero mean, or high frequency components. For the zero and low frequency components, a scaling function, which has unit mean value, is required. The wavelet and the scaling functions with the dyadic scaling and translation parameters build a complete orthogonal basis in the Hilbert space.

The precise mathematical description of orthonormality is easily found in almost every book about the wavelet transform, e.g. [1] and [2], and will not be repeated here.

From this definition it is known that the wavelet transform is the integral of the multiplication of the signal to be studied,  $f$ , with a wavelet function,  $\psi$ . It has the same form as the STFT. However, not like the STFT, where only sine and cosine functions are used for the transformation, the wavelet transform uses different wavelet functions, which can be se-

lected according to the specific problems from a principally unlimited set.

Parameter  $a$  defines the width and height of the wavelet function  $\psi$  keeping its unit square norm. If  $a$  makes  $\psi$  narrower, the wavelet represents fast changes and the transformation focuses on the high frequency components of the signal. Parameter,  $b$ , shifts the wavelet function along the time axis, so that the transform helps us to observe signals at different locations. Using different values  $a$  and  $b$ , it is possible to observe the signal at different positions in the time domain and in different frequency ranges with only one transformation.

Two forms of wavelet transform are available: Continuous wavelet transform (CWT) and discrete wavelet transform (DWT). In CWT both scale and position parameters are continuous real values. The transformation result, i.e. the wavelet coefficient, is therefore also continuous. CWT expresses the signal changes in a continuous manner. It is more suitable for the visual examination. However, the transform result contains redundant information and the transform requires large calculation efforts. The general CWT has no corresponding inverse transformation.

## 2.2 Discrete wavelet transform (DWT)

In the DWT only discrete values of the scale and location parameters are used. The values are determined according to point (4) of the wavelet properties. The transformation result, i.e. the wavelet coefficient, is therefore discrete.

As an example, the following figure shows the form of the third order Daubechies scaling and wavelet functions [1].

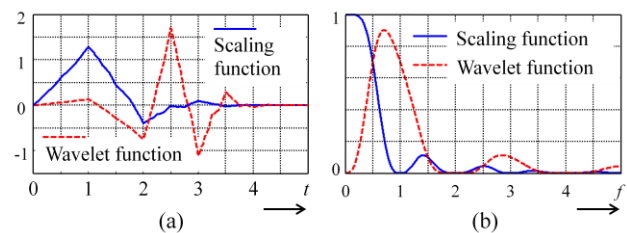


Figure 1: The third order Daubechies scaling and wavelet functions (a) with their Fourier transforms (b)

From the Fourier transforms it can be seen that the scaling function mainly covers the lower frequency range while the wavelet function stretches in a higher frequency range. This coincides with the aforementioned description. From this point of view, DWT is actually the division of the time signal into different frequency bands. Thus, it is straightforward to understand that the calculation of DWT is realized

using filter banks. In inverse DWT the calculation is similar. This process can be illustrated with Figure 2.

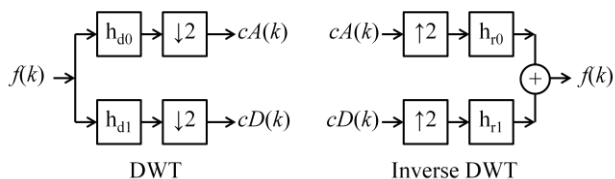


Figure 2: DWT and inverse DWT calculation using filter banks

DWT transforms the original sequence in two new series:

- (1) the approximation coefficients,  $cA(k)$ , representing the low frequency components, obtained using the low pass filter for decomposition,  $h_{d0}$ , and
- (2) the detail coefficients,  $cD(k)$ , representing the high frequency components, obtained using the high pass filter for decomposition,  $h_{d1}$ .

The symbol  $\downarrow 2$  means down-sampling. The operation is to delete one from every two adjacent coefficients, in order to remove the redundant information. The inverse DWT carries out the reversed operation. The operator,  $\uparrow 2$ , expands a coefficient series by inserting a zero between every two adjacent elements. After that the two series are operated with two filters, respectively, and added together to get the original signal.

### 2.3 Multi-resolution analysis

Considering the DWT process shown in Figure 2, sequence,  $cA(k)$ , which represents the low frequency components, can be further divided into a lower frequency part and a higher frequency part. This process can be repeated and a series of coefficient sequences representing different frequency ranges will be obtained, as shown in the following figure:

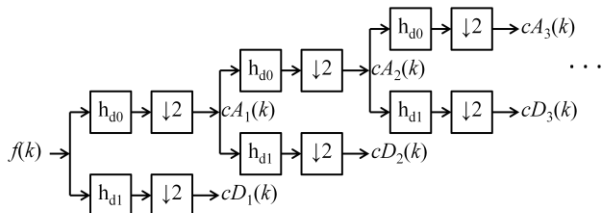


Figure 3: Wavelet multi-level decomposition

This process is named as wavelet multi-level decomposition. The output of this operation,  $cD_1, cD_2, \dots, cD_n$  and  $cA_n$ , are different levels of DWT coefficients, representing the signal components from higher to lower frequencies. This analysis provides a convenient tool to observe different frequency components of the signal depending on time.

Similar to the calculation process of inverse DWT, if the algorithm goes in an inverse direction of Figure 3, we will get the original signal using the wavelet coefficients of different levels. This process is called signal reconstruction.

Applying multi-level decomposition, we can project a signal into a sequence of nested subspaces. We are then able to observe the original signal in different subspaces that contain different details of the signal. This realizes the multi-resolution analysis (MRA) of the signal using wavelet transform.

## 3 Realization

The wavelet library is developed according to the Modelica Language Specification 3.2 with the simulation environment software, Dymola 2013 (32-bit) demo version. Although the Dymola demo version has limitations on the complexity of simulation models, it does not impose any limitations in programming.

### 3.1 Library structure

The wavelet library structure is sketched in Figure 4.

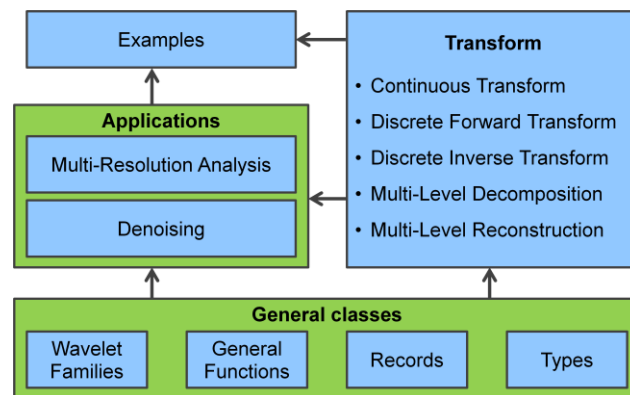


Figure 4: Wavelet library structure

The whole library is composed of eight packages and 64 single classes, plus an external C-file to support generating Meyer wavelets. All of these elements are summarized in four parts as shown in the figure.

The central part of this library is the package “Transform”, which includes the functions carrying out different wavelet transformations.

The part “General classes” provides a fundament for realizing the wavelet transforms and other operations, as shown in the bottom part of the above figure. Four packages are included in this part. They are

- “Wavelet Families” for generating wavelet filter banks and functions,
- “General Functions” for the basic data operation and processing,
- “Records” for the definition of data structures and supporting GUI, and
- “Types” to define enumeration parameters.

Two wavelet applications – MRA and denoising – are provided. It should be noted that the wavelet denoising package can also be applied for the data compression since these two operations use the same concept and algorithm [2].

In addition, the package “Examples” consists of functions and models for six examples to provide the users a quick introduction to the wavelet library.

### 3.2 Wavelet library release

The functional part of this library consists of only two files:

- “Wavelet.mo” as the main library package to include all Modelica codes, and
- “fft\_c.c” as an external C-function to calculate Fourier transform for generating Meyer wavelets.

Two example data files, testSignal1.mat and testSignal2.mat, are delivered, although they could be very easily generated by running the two example models in the library.

In addition, a “Help” folder including all descriptions of this library is also a part of the delivery.

### 3.3 Required components

Besides the Modelica standard libraries, the full performance of this library relies on two further libraries, **Modelica\_LinearSystem2** and **Plot3D**. The former one is a free library available from Modelica Association. The latter one is delivered with Dymola, and is only used for showing CWT results. In the free demo version of Dymola, Plot3D works but is limited to two-dimensional images.

### 3.4 Wavelet families

Fifteen wavelet families are available in this library.

- Haar
- Daubechies , up to the 20-th order
- Symlets , up to the 20-th order
- Coiflets , up to the 5-th order
- Biorthogonal spline, 15 variations
- Reverse biorthogonal spline, 15 variations
- Discrete Meyer
- Meyer

- Gaussian , up to the 8-th order
- Mexican hat
- Morlet
- Complex Gaussian , up to the 8-th order
- Complex Morlet
- Complex Shannon
- Complex frequency B-Spline , unlimited order

The package “Families” consists of fifteen functions for the wavelets listed above. Two more functions that are directly related to wavelet families are also included: “scalingWaveFunc” for generating the wavelet and scaling functions of a specific wavelet, and “wavFunc” providing a common entry to access all fifteen wavelets.

### 3.5 Wavelet transform

Five variations of one-dimensional wavelet transform have been implemented in the library. The related functions are included in the package “Transform”.

CWT is provided with two variations for more flexibility. One CWT variation accepts a wavelet name while the other one accepts a wavelet function as the input parameter. According to the wavelet theory, no inverse transform exists for general CWT.

For the discrete version, both forward and inverse wavelet transforms are possible. They are realized with two Modelica functions, respectively.

In the discrete domain, the multi-level data decomposition and the reconstruction are carried out by actually applying the discrete wavelet transform and inverse transform in a cascaded manner. The algorithms are implemented by two Modelica functions, “wavDec” and “wavRec”, respectively.

In addition, two more functions are included in this package, too, since they are closely related to the transforms. Function “wavRec1” reconstructs the data using only the wavelet coefficients of one specific level. This is actually a subset of multi-level reconstruction. The other function “wavCoef1” is used to extract the wavelet coefficients of a certain decomposition level. These two operations are specially useful for wavelet applications, such as MRA and denoising.

### 3.6 Applications

The wavelet library provides two applications, the MRA and the denoising, which are supposed to be the mostly used functionalities for processing the simulation data.

The principle of the wavelet MRA has been described in section 2.3. With this tool the user is able to divide the signal into different scales, or frequency regions, and observe the signal with different resolution. Using different Modelica functions, users can get the MRA results either in the numeric form or in the graphic form with curves. In addition, this tool provides the flexibility to tune the coefficients of every single level, so that the user can intentionally strengthen or suppress the information in some certain levels. An example of MRA is to be given later in this article.

Like MRA, the wavelet denoising is carried out based on wavelet multi-level decomposition and reconstruction. The wavelet denoising is suitable only for data with high signal-to-noise ratio. Based on this condition, the wavelet coefficients representing the noise have smaller values compared with those representing useful information. By removing these small coefficients, the noise can be eliminated from the data. The threshold to separate the noise and the information coefficients is the key to this algorithm. It can be selected by the user or automatically estimated by the tool. It has to be noted that, if automatic estimation is used, the noise in the data must have Gaussian distribution with zero mean and unit standard deviation.

As an extra possibility, data compression can be realized by the denoising application tool since these two operations are actually based on the same algorithm. For more information about this topic, relevant literatures, such as [1] and [2], can be referred to.

### 3.7 Graphical user interface

A graphical user interface (GUI) is important for a user-friendly environment in this library. The GUI is mainly realized by using Modelica “Types” combining the simulation environment software for inputs and diagrams for outputs.

By combining the GUIs for input and output, a very simple but convenient interactive operation is possible. It is also possible to realize more complex and user-friendly interactive performance using specific scripts written by users.

### 3.8 User’s guide

A complete User’s Guide in PDF format with a detailed description of every single class in this library is delivered in the release version. In addition, a “Help” folder is included in the delivery package. This folder contains HTML files with hyper-links.

Within the library, no separate package serving as a User’s Guide is provided since all related descriptions about the Modelica classes, including Packages, Models, Functions, Types and Records, have already been embedded in the library. These descriptions can be easily accessed in the simulation environment software. Hyper-links are included in the description texts for quick movement among the library to access the information from different classes.

## 4 Testing the library

The algorithms of this wavelet library were completely tested with the black-box method. Testing vectors have been defined for each algorithm. The function outputs were checked visually and numerically. The numeric check was carried out with the assistance of MATLAB because most algorithms written in the Modelica wavelet library are included in the MATLAB wavelet toolbox. For these algorithms, the same input parameters were given to the functions of the Modelica wavelet library and the corresponding commands in MATLAB. The calculation results of both tools were compared. An algorithm of the library is considered correct if the discrepancy is below a certain limit.

All testing results were recorded and summarized. As an example, Table 1 shows the summary of the testing results of the function “Wavelet.Families.wavCoiflets”, which is used to generate the wavelet filters associated with Coiflets wavelets.

Table 1: Testing recording for Wavelet.Families.wavCoiflets

order	F	lod	hid	lor	hir	Result
1	-	-	-	-	-	Error = 0
2	-	-	-	-	-	Error = 0
3	-	-	-	-	-	Max. error < 1e-16
4	-	-	-	-	-	Max. error < 1e-16
5	-	-	-	-	-	Error = 0

The first six columns of the table records the input and output values of the function. Since the data amount of the output values is too large, the data are stored in a separate file. The last column “Result”, records the errors of the outputs obtained by the Modelica function under test compared with the data calculated by the same algorithm written in MATLAB.

All other functions in the library were tested with the same process and altogether 133 numeric testing results were recorded. The errors are illustrated in Figure 5. Since the values are shown in logarithm axis,

the first 58 points with zero value are not plotted. The testing showed that the more than half of the tests have the errors below  $2e-16$ , which is around the digital error of a double precision floating number. Other errors with the values up to  $5e-14$  might be caused by the precision of the direct constant numbers and difference in details of the coding in the algorithms.

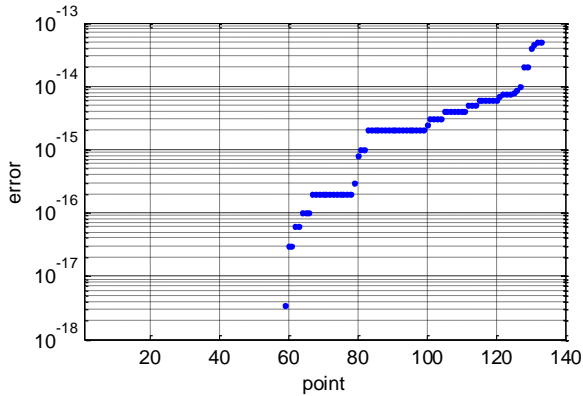


Figure 5: Error distribution of the testing results

In addition, visual checks were carried out for all functions that have graphical inputs and outputs. All results coincided with the specifications. For the functions outputting curves and images, the graphic data are actually numerical results and also tested numerically.

## 5 Examples

In this section, we take two examples to partially show the usage and functionalities of the new Modelica Wavelet Library in the simulation environment, Dymola.

### 5.1 The signal under study

The signal under study is defined in equation (2).

$$\begin{aligned}
 y &= y1 + y2 \\
 y1 &= \begin{cases} \sin(2\pi 10t), & t < 0.5 \\ 0.2 \sin(2\pi 10t), & t \geq 0.5 \end{cases} \\
 y2 &= \begin{cases} 0.2 \sin(2\pi 50t), & t < 0.5 \\ \sin(2\pi 50t), & t \geq 0.5 \end{cases}
 \end{aligned} \quad (2)$$

It is a time-variant signal containing two frequencies, 10 and 50 Hz. At time  $t = 0.5$  s, the magnitudes of the two frequency components alternate. Signal  $y$  is generated with the Modelica model “testSignal2” in the package “Examples” of the wavelet library. It is shown in Figure 6.

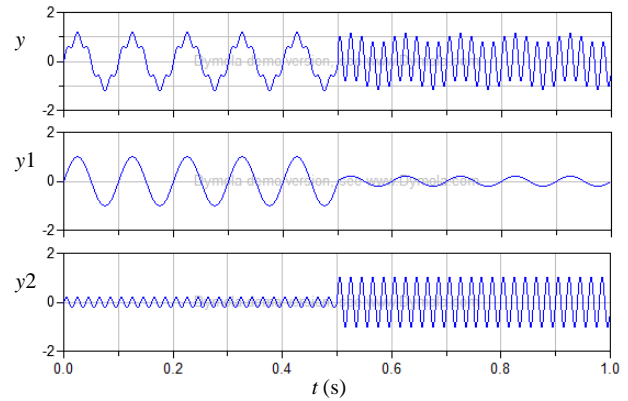


Figure 6: The signal under study for the examples

Since the wavelet transform can only be correctly performed with an equidistant time grid, the simulation data should be stored with an equidistant time grid. Otherwise, the data have to be converted to equidistant time grid with the library function “interpL” before applying wavelet transform. In the examples, this is ensured by setting the sampling frequency “fs” to a none-zero positive value.

### 5.2 Wavelet multi-resolution analysis

Firstly we carry out an MRA using the wavelet library to observe the signal around the transient time,  $t = 0.5$  s with different resolutions. This is done by executing the function “fileDataMRA” in the package “Examples”. Most of the input parameters can be left at default values except the following ones:

- Time range for analysis is set as  $t_0 = 0.25$  and  $t_1 = 0.75$  since we are only interested in the transient region;
- Sampling frequency  $fs = 480$ , so that the highest frequency in the testing data is 240 Hz ( $fs/2$ );
- $Nd = 7$  in the wavelet definition “wd” to select the 7-th order Daubechies wavelet;
- $decLevel = 3$  in the MRA parameters “mraParameters” for carrying out a three level wavelet decomposition; and
- $rA = 0$  in “mraParameters” to remove the lowest frequency components in the reconstructed signal.

After execution, we get eight curves displayed in two diagrams in Dymola. In the first diagram, the original signal and the data in the approximation level and three detail levels are displayed. They are shown here in Figure 7.



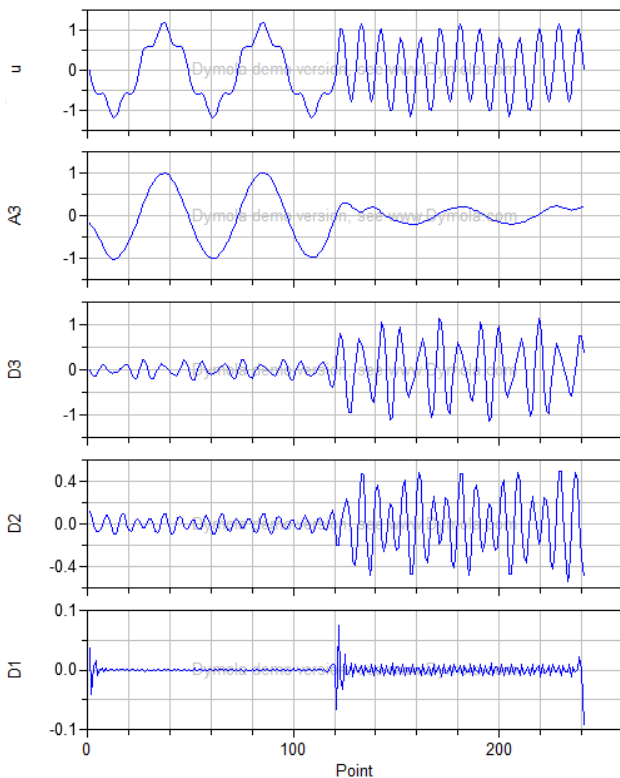


Figure 7: The original signal and decomposed data

The x-axis of the diagram shown in Dymola is given in data point. There are 240 data points; and the symbol of the original data is shown as “u” instead of “y”.

The wavelet decomposition in this MRA separates the original signal into four time series containing different frequency components. This is illustrated with Figure 8.

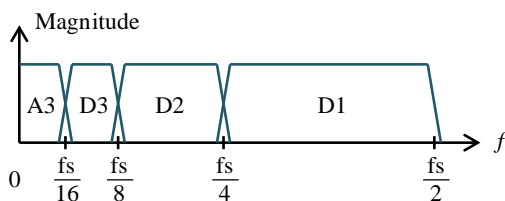


Figure 8: Separation of the frequency components with three-level wavelet multi-resolution analysis

Since the sampling frequency is  $f_s = 480$  Hz, we know that the 10 Hz component of the original signal is projected into level A3 and the 50 Hz component falls in D3. From Figure 7 we also see some oscillations in level D2. This is because the wavelet decomposition does not realize perfect frequency separation. Furthermore, some peaks in level D1 represent the frequency transient and edge effect, which imply fast changing events.

The second half of the MRA in this example is data reconstruction after the tuning of the wavelet coefficients. This is shown with four curves in the second

diagram, which is not repeated in this article. Since we have set  $rA = 0$ , meaning to remove the information in the approximation level, which mainly contains the 10 Hz component, the reconstructed data mainly contains the 50 Hz component, as shown Figure 9. We can see that this signal is almost identical to the middle part (0.25 – 0.75 s) of signal y2 in Figure 6.

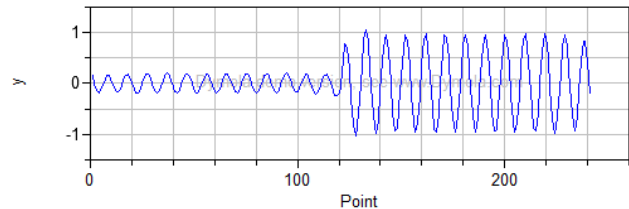


Figure 9: The reconstructed data after removing the approximation coefficients in wavelet MRA

### 5.3 Continuous Wavelet Transform

The second example demonstrates CWT for the same signal as shown in Figure 6. By executing the function “fileData\_cwtn” without changing any default parameters we can get the CWT result illustrated with a 2-dimensional image or a 3-dimensional surface, depending on the version of Dymola used. The image with pseudo-colour is shown in Figure 10.

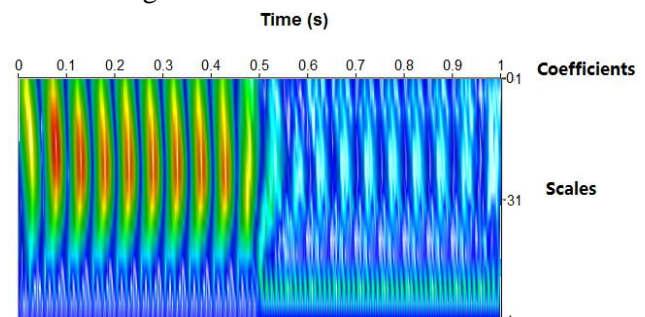


Figure 10: Continuous wavelet transform of the signal under study

The coefficients are shown with warmer colours (yellow-orange-red) for larger magnitudes and colder colours (green-light blue-dark blue) for smaller magnitudes. The horizontal axis represents the time from 0 to 1 second. The vertical axis represents the wavelet scales from 1 to 64. The relationship between scales and frequencies is determined by both the sampling frequency of the data and the wavelet used. By observing this CWT image we can draw several conclusions:

- The frequency transient happens at  $t = 0.5$  s.
- Both low frequency components (here 10 Hz) and high frequency components (here 50 Hz) are present throughout the time span of the signal.

- Before 0.5 s low frequency components are much stronger than the high frequency components. This reverses after 0.5 s.

## 6 Conclusion

This article describes a new wavelet library developed for Modelica. This library contains fifteen wavelet families commonly used in research and engineering applications, continuous wavelet transform, discrete wavelet transform both in forward and inverse directions, two application tools and several examples. In addition, the library provides graphic user interfaces for some functions to support a user-friendly workflow. Moreover, it includes documentation embedded in the library, plus an external user's manual.

During and after the development work, black-box testing was carried out for all algorithms. The testing results have proved the correctness of the library.

The first release of this Modelica wavelet library is limited to one-dimensional wavelet transforms for post-processing of the simulation data. Several wavelet calculations, such as wavelet packet and two-dimensional transform, are not yet included. This could be gradually added in the future. Nevertheless, the introduction of this wavelet library will definitely enrich the functionality of Modelica and contribute much to the Modelica society.

## Acknowledgement

The authors thank the support from the Clean-Sky Joint Undertaking through the Grant Agreement No. 296369 (Project MoMoLib) within the Seventh Framework Programme of the European Union.

## References

- [1] I. Daubechies. Ten Lectures on Wavelets. SIAM 1992.
- [2] S. Mallat (2009): A wavelet tour of signal processing - the sparse way. Amsterdam: Elsevier.
- [3] X. Zeng, S. Huang, et. al. Frequency domain wavelet method for high-speed circuit simulation. IEEE International Symposium on Circuits and Systems. ISCAS 2002, pp II-233-236, 2002.

- [4] Q. Wang and F. Jiang. Simulation for fault of transmission lines based on discrete wavelet transform. International Conference on Electrical and Control Engineering. ICECE 2011, pp 2554-2556, 16-18 Sept. 2011.
- [5] G. Ala, M.L. Di Silvestre, et. al. Wavelet-based efficient simulation of electromagnetic transients in a lightning protection system. IEEE Transactions on Magnetics. ITM 2003, vol 39, no 3, pp 1257-1260, May 2003.
- [6] W. Xu, Y. Cui and J. Liu. Simulation Study of Detecting Pulse Signal Based on Wavelet transform. 8th International Conference on Electronic Measurement and Instruments, 2007. ICEMI '07, pp II-479-483, Aug. 16-18 2007.
- [7] T. Bunte, A. Sahin and N. Bajcinca. Inversion of Vehicle Steering Dynamics with Modelica / Dymola. Proceedings of the 4<sup>th</sup> International Modelica Conference, pp 319-328. Hamburg, Germany, March 7-8, 2005.
- [8] Y. Ji and J. Bals. A Modelica signal analysis tool towards design of More Electric Aircraft. Proceedings of 2010 International Conference on Information and Applied Electronics, pp 152 - 156. Chendu, China, June 9-11 2010.
- [9] J. Gao, Y. Ji, J. Bals and R. Kennel. Fault Detection of Power Electronic Circuit using Wavelet Analysis in Modelica. Proceedings of the 9<sup>th</sup> International Modelica Conference, pp 513 - 521. Munich, Germany, Sept. 3-5, 2012.
- [10] E. Jacobsen and R. Lyons. The sliding DFT. Signal Processing Magazine, vol. 20, issue 2, pp. 74-80, March 2003.

# Model-based Verification and Optimization of Batteries for Mobile Power Applications

Marco Franke, Tamas Juhasz, Ulrich Schmucker  
Fraunhofer Institute for Factory Operation and Automation IFF  
Sandtorstr. 22, 39106 Magdeburg, Germany  
{marco.franke, tamas.juhasz, ulrich.schmucker}@iff.fraunhofer.de

## Abstract

This paper describes a new approach to sizing an electric drivetrain, including its power supply. The advantages of a real battery testing system are combined with the advantages of a Modelica model of a product. A testing system analyzes battery performance under specific constraints such as differing temperatures, vibrations and humidity. Since these constraints are hard to replicate in a model, an experimental rig is an optimal solution for battery tests [1]. Virtual engineering of a real system is advantageous in terms of the speed of modifications, the measurement of all relevant data and the low cost of development. The coupled virtual model and experimental testing rig for batteries constitute the basis for this new concept and improve the development of the electric powertrain.

*Keywords: battery, hardware-in-the-loop, real-time*

## 1 Introduction

Mobile applications are increasingly using electric power supplies, which have to be sized optimally. Electric vehicles are stimulating the development and optimization of future drivetrains. Their physical and temporal range is very important.

This study focuses on one of the main components, the power supply. Batteries are normally tested in test rig environments. The advantages of an experimental rig are its versatility, e.g. for climate tests, vibration tests and discharge tests. One of its most important features is its safety for operators and the environment. Tests of batteries installed in real vehicles or equipment involve various safety issues and unforeseen conditions or defects can cause injuries or other safety-critical situations.

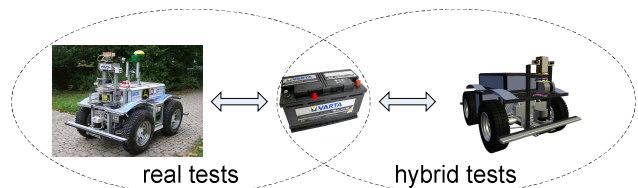


Figure 1: Hardware-in-the-loop with a real battery

Since current battery models are often incomplete or only describe properties in certain areas, the advantages and potentials of real test environments also ought to be factored into the modeling of mobile systems and, specifically into the modeling of electric drivetrains.

## 2 The Experimental Vehicle RAVON

Since detailed data from vehicle manufactures was unavailable for the development of the methodology presented here, the AWD off-road vehicle RAVON (Robust Autonomous Vehicle for Off-road Navigation) developed by the Technical University of Kaiserslautern, Germany was used as the test subject.



Figure 2: The autonomous vehicle RAVON

This unmanned electric vehicle is normally used as a test object to study autonomous, behavior-based strategies for motion adaptation, localization and navigation in rough outdoor terrain. Equipped with four synchronous wheel hub motors and four batteries, the vehicle can be used under a variety of temperature, humidity and terrain conditions. The electric vehicle is powered by eight 12V batteries with a total voltage of 48V. This power supply must be adequate for the entire task.

Since the data needed to create a mechatronic model of RAVON (e.g. CAD data, drivetrain and control parameters and battery specifications) were available in advance, modeling with Modelica could start immediately.

### 3 The Modeling Procedure

The RAVON's CAD data are available in STEP format. These data are the basis for the mechanical parts of the chassis with interfaces to other physical domains in Modelica. The Fraunhofer Institute for Factory Operation and Automation IFF in Magdeburg has an automated CAD-to-simulation procedure that effectively converts CAD data into Modelica description [2].

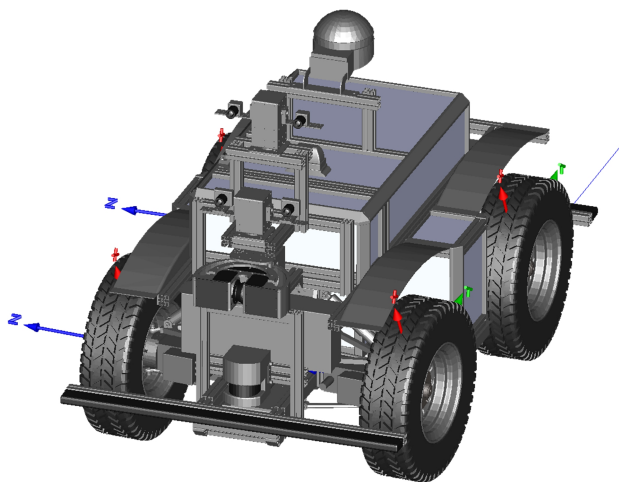


Figure 3: Modelica model generated semi-automatically from CAD data

The program's database contains XML data on the mechanical model's physical and kinematic properties (masses, moments of inertia, joint constraints, etc.) as well as triangulated VRML geometry exported directly from the CAD system. Since it also supports closed kinematic loops, this new procedure also makes it possible to model any complex mechanical system efficiently.

### 3.1 RAVON Subsystems

The virtual RAVON platform must communicate with a real battery test rig in real time. Apart from incorporating the semi-automatically generated chassis model, the most demanding task was modeling the entire electric drivetrain so that it is real-time capable. The uppermost vehicle model was decomposed into two Modelica subsystems, *Chassis* and *Drive* (see Figure 4).

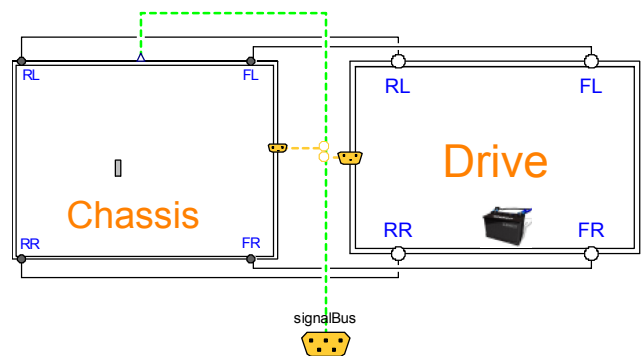


Figure 4: Modeled subsystems of RAVON

The virtual parts of the *Drive* model can be matched with their real counterparts. A *Huebner HAC 71.11.6* permanent magnet synchronous induction machine and a *CycloDrive 6000* planetary gear are mounted in each wheel-hub model. The electrical machines were parameterized in Modelica using real manufacturer data. A DC interface to a common external power supply was also included.

The drive subsystem of a single wheel with the DC/AC converter and the motor controller is pictured in Figure 5.

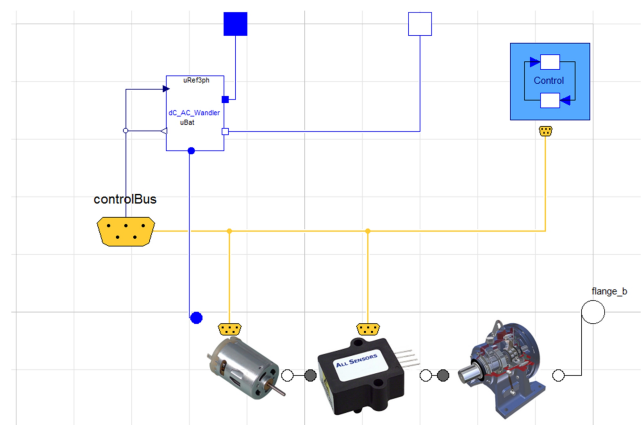


Figure 5: Controlled wheel-hub motor model

The motor controller model is based on a conventional cascade structure with an outer speed control loop and an inner current control loop. PI controllers with limited outputs are used in both cases. Since the real-time capability had top priority, some simplifications were made in the model of the motors' power electronics, among others. Since they would not influence the model's accuracy significantly, fast PWM signals were not included. The torque control algorithm is based on field-oriented vector control [3]. The controller parameters were selected to attain maximum speed with minimal overshoot.

## 4 Battery Testing System

This study aimed to utilize a real battery testing system in order to integrate a real battery's complex behavior in early stages of product development. Rather than using a real product prototype to record battery charging profiles, a virtual prototype is incorporated in the workflow. A real testing rig can be used to swap energy bidirectionally between the virtual model and the real battery, depending on the prevailing conditions.



Figure 6: FuelCon battery testing system

Since the Modelica model provides the load for a real battery, its performance can be analyzed dynamically. The testing system from the company FuelCon allows users to set different environmental parameters such as ambient pressure, temperature, atmospheric humidity and movement. Various types of batteries, e.g. lithium-ion, lithium-polymer, NiCd, NiMH or lead-acid, can be tested with this testing system. The Evaluator-B series battery testing system has an interface that is connected with external current profiles [4]. The methodology developed here with the virtual vehicle model uses this function to repeatedly supply actual current and receive the battery voltage.

## 5 Integration of the Real Testing System in a Hybrid Simulation

The objective of this study was to integrate a real battery testing environment in a hardware-in-the-loop simulation setting. Battery voltage and computed electric current signals must be exchanged bidirectionally during a test run. The testing system is equipped with load and charger electronics. This makes it possible to charge or discharge the batteries depending on prevailing electric currents. Dynamic processes such as current peaks can also be analyzed with this approach.

### 5.1 Software Interfaces

The FuelCon testing system's operator interface utilizes Visual Basic (VB) scripts running in an embedded Windows environment. The Fraunhofer IFF developed the COMsigate interface to extend the operator interface's range of functions. The COMsigate software module employs the User Datagram Protocol (UDP) for communication between the testing system and the outside world.

The main VB script runs in a loop during battery-coupled hardware-in-the-loop tests. It reads the actual battery voltage ( $U$ ) and can set a new electric load ( $I$ ) every 10ms in keeping with the latest UDP packet received through the COMsigate.

Matlab/Simulink was incorporated in the tool chain because of its excellent interoperability and advanced toolboxes [5]. Its UDP communication blocks were used to transmit the battery signals  $U$  and  $I$  bidirectionally through the COMsigate interface. In addition autonomous driving course profiles, the vehicle model can also be interactively controlled with a joystick module that enables users to define own test cycles in Matlab.

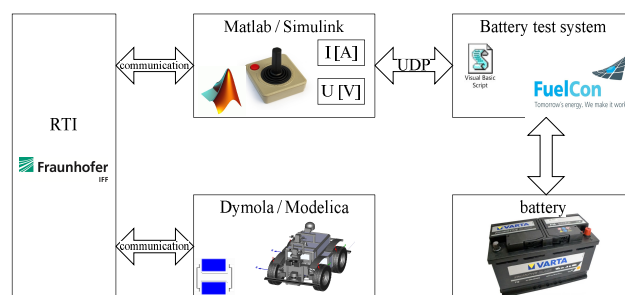


Figure 7: Tool chain of the hybrid simulation of RAVON with a real battery in a test rig

The Fraunhofer IFF developed the Real Time Interface (RTI) to support real-time communication between heterogeneous software applications [6]. RTI's shared memory is used to exchange battery charge states and reference speed signals between Matlab/Simulink and the Dymola simulator [7] running the RAVON model.

The reference vehicle speed and the axles' steering movement are supplied to Dymola. Each electric drive is controlled separately. The vehicle's acceleration is determined by the electric current in the respective motors. The virtual supply voltage is the value of the real battery measured in the test rig. Thus, the real battery's performance influences the simulated vehicle's electric power.

## 6 Results

The first tests revealed that the dynamic processes between the battery testing system and the vehicle model reflect the theoretical considerations. The first example (Figure 8) presents a battery performance test using a static power consumption scenario. The model vehicle had to travel at a constant reference speed. Decreasing voltage increases the demand for electric current as the battery discharge. Since the mechanical power demanded is constant this phenomenon can be explained by the electric power balance equation:  $P_{elec} = U \cdot I$ .

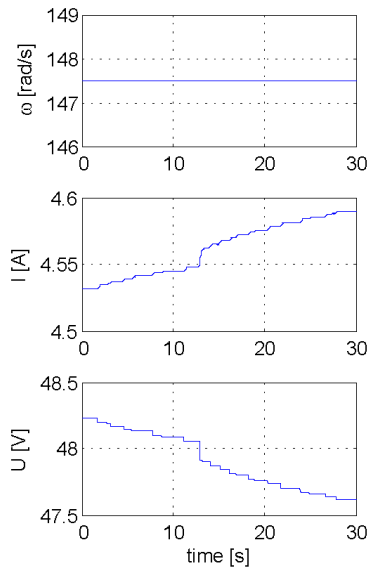


Figure 8: Discharging at constant travel speed

The actual speed remains constant until the maximum current is reached, which is defined by the battery. From this point onward, the vehicle's speed cannot be kept constant and decreases. Afterward,

the electric current reaches a constant value and the battery voltage continues to fall.

The results presented in Figure 9 depict forward and backward acceleration and deceleration maneuvers with sudden stops. The battery's demand for current increases during the acceleration phase and the battery's voltage drops temporarily. The current peak when the electric motor switches from loading to braking is evident. Such current peaks during charging and discharging can affect a battery's life and also influence a battery's state of charge.

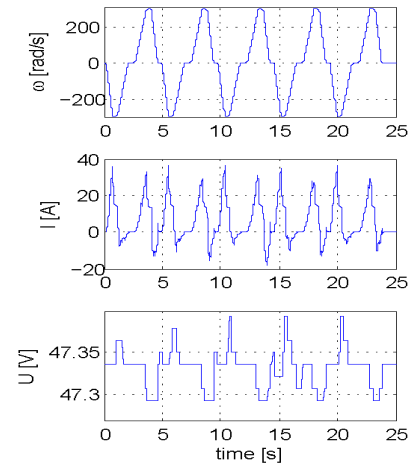


Figure 9: Analysis of dynamic behavior

A continuous and rapid change of direction was investigated in the next test scenario. Electric current and voltage propagation were analyzed. Figure shows the rate of speed as well as the related current and voltage.

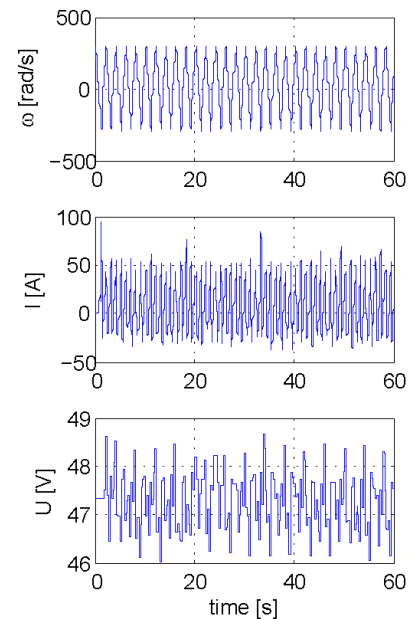


Figure 10: Continuous and rapid changes of direction

The current peaks in the case presented here are many times higher than those in the test scenario in Figure 9.

Verlagsschriftenreihe. Bd. 252. Paderborn, 2009 — ISBN 978-3-939350-71-2, pp. 155–170

[7] Dassault Systèmes - <http://www.3ds.com/>

## 7 Conclusions

This paper describes a novel hardware-in-the-loop approach to connecting mobile mechatronic system models with a real battery testing system. It can be used to test the entire system's energy balance under realistic conditions. This makes it possible to perform stress and durability tests without having to build complex prototypes first. Battery size and the product's electromechanical properties can be optimized. As a result, product development is expedited and validated by early testing. This approach is suitable for more than just electric vehicles. It can also be applied to all types of battery-powered devices with higher power requirements.

## Acknowledgement

The here presented work has been funded by the German Federal Ministry of Education and Research within the joint project “ViERforES-II - Virtual and Augmented Reality for Maximum Embedded System Safety, Security and Reliability” (01IM10002A) of the program IKT2020.

## References

- [1] Benecke, I.; Schmucker, U.: “Simulation statt Prüfstand” in Quality Engineering Vol 04.13, 2013, ISSN 1436-2457, pp. 40-42.
- [2] Juhasz, T.; Schmucker, U.: “Automatic Model Conversion to Modelica for Dymola-based Mechatronic Simulation”; Proc. of 6th International Modelica Conference, 3rd-4th March, 2008, Bielefeld, Germany, Vol. 2, pp. 719-726.
- [3] Schröder, D.: “Elektrische Antriebe - Regelung von Antriebssystemen“; Springer Verlag 2009; ISBN 978-3-540-89612-8
- [4] FuelCon AG - <http://www.fuelcon.com/>
- [5] MathWorks - <http://www.mathworks.de/>
- [6] Böhme, T.; Kennel, M.; Schumann, M.; Winge, A.: “Automatisierte Erstellung domänenübergreifender Modelle und echtzeitfähige Kopplung von Simulation, Visualisierung und realen Steuerungen”, HNI-





# Systems Physics Library

Werner Maurer  
Institut für Angewandte Mathematik und Physik  
Zürcher Hochschule für Angewandte Wissenschaften  
Technikumstrasse 9, Winterthur, 8401 Switzerland  
[maur@zhaw.ch](mailto:maur@zhaw.ch)

Elisabeth Dumont  
Institut für Angewandte Mathematik und Physik  
Zürcher Hochschule für Angewandte Wissenschaften  
Technikumstrasse 9, Winterthur, 8401 Switzerland  
[dumo@zhaw.ch](mailto:dumo@zhaw.ch)

## Abstract

In this poster the *Modelica Systems Physics library* is presented. The Systems Physics library is a free open-source library with models for different areas of physics [1]. The primary use of the library is for educational purpose in Physics courses at medium level.

The library contains models from five different domains (hydraulics, translational and rotational mechanics, electrodynamics and thermodynamics). In the future, we plan to add chemistry as a sixth domain. Each domain contains connectors containing a substance-like quantity and the corresponding potential; basic models (capacitance and resistance); sensors and actuators as well as some domain specific elements, such as nonlinear accumulator, nonlinear resistors, valves, springs or inductances. In addition to the constitutive equations each model also comprises the energy balance. For example, dissipative elements calculate the loss energy and even the entropy production with the help of an additional thermodynamic connector.

*Keywords: physics; education; system dynamics*

## 1 Introduction

Systems Physics is a novel approach to physics with which beginners are able to grasp the fundamental concepts underlying processes in nature and technology [2]. It is based on everyday concepts known from fluids which are familiar to everybody. The analogy between physical quantities and fluids offers a very intuitive approach to physics [3]. The powerful pictorial modeling offered by Modelica helps students to understand basic physical processes.

Moreover, here is an immense number of problems that can be addressed by this approach which are usually not included in the standard physics textbooks at undergraduate level. With the help of this Modelica library, students are able to model rather

complicated physical systems (e.g. friction) with little mathematical knowledge.

In the first two semesters at the ZHAW School of Engineering students learn the modeling concept of system dynamics with Stella or Berkeley Madonna. Thereby they learn the basic structures and formulas (balance equation, constitutive laws and the energy carrier concept) of physics. The *Systems Physics* library could then be used in subsequent semesters in order to deepen this knowledge.

Systems Physics combines the modeling concept of System Dynamics with a unified description for all branches of classical physics known from Bond Graph theory [4]. Our concept of energy carrier is similar to that of the Bond Graph theory. But there is a crucial difference. In Bond Graph theory, force and torque are potential quantities (effort quantities) and the kinematic variables velocity and angular velocity are seen as flow quantities. In Systems Physics however, this approach is not possible because force and torque are part of the balance equation and velocity and angular velocity are the “driving force” for the appropriate currents. Therefore force and torque are flow variables and the two velocities are effort or potential quantities.

In our model based approach students start by formulating the balance of a fundamental quantity (bathtub-thinking for volume, mass, electrical charge, momentum, angular momentum, entropy or amount of substance). Then they have to specify the currents and the rates of change (feedback-thinking). On a second layer they can add the balance of energy (bookkeeper-thinking).

Our vision of a new physics course for engineers or natural and medical scientists covers system dynamics modeling in the first year and object-oriented modeling in the following years of studies [5].

## 2 Structure

The systems physics library includes the domains hydraulics, electrodynamics, translational mechanics, rotational mechanics, thermodynamics and chemistry. Each domain includes a substance like quantity and a corresponding potential. All connectors are constructed according to this basic structure (Table 1)

In all branches of physics there are many different storage systems such as tanks, plastic bottles, capacitors, moving or rotating bodies, heat accumulators. For each storage system we can write down two balance equations: one for the basic quantity  $M$  (volume, electric charge, momentum, angular momentum, entropy); and one for the Energy  $W$ . The balance equations for  $M$  and  $W$  are given by

$$\sum_i I_{M_i} = \frac{dM}{dt} \quad (1)$$

$$\sum_i I_{W_i} = \frac{dW}{dt} \quad (2)$$

where  $I_M$  are substance currents and  $I_W$  are energy currents. For the entropy balance equation a production rate  $\Pi_S$  is introduced in addition to entropy currents.

The domain specific potential  $\varphi_M$  (pressure, voltage, velocity, angular velocity and absolute temperature) connects the current for the basic quantity with the energy flux

$$I_W = \varphi_M I_M \quad (3)$$

A Model with two ports and a conserved quantity (ideal volume, electric charge, momentum or angular momentum) contains at least one equation for conservation and one for the power

$$I_{M1} + I_{M2} = 0 \quad (4)$$

$$P = \Delta \varphi_M I_M \quad (5)$$

These equations are formulated in separate *partial models*. In thermodynamics we need two different partial models, one for heat conductance which conserves energy

$$T_1 I_{S1} + T_2 I_{S2} = 0 \quad (6)$$

and one for ideal heat engines which conserve entropy

$$I_{S1} + I_{S2} = 0 \quad (7)$$

An additional equation calculate the production of entropy in the partial model for heat conductance

$$I_{S1} + I_{S2} = \Pi_S \quad (8)$$

or the power in a partial model for ideal heat engines

$$P = \Delta T I_{S1} \quad (9)$$

Ideal heat pumps are modeled similar to hydraulic pumps: in the same way as a hydraulic pump pumps water or oil heat pumps pump entropy. This is one of the main messages of Systems Physics.

Table 1: flow and potential variables in the different domains of Systems Physics

domain	quantity	potential
hydraulics	volume	pressure
electrodynamics	charge	voltage
translational mechanics	momentum	velocity
rotational mechanics	angular momentum	angular velocity
thermodynamics	entropy	temperature

### 2.1 Hydraulics

The volume of an incompressible fluid is the basic quantity in hydraulics and pressure is the associated potential. Because pressure multiplied by volume flow equals the flux of energy, the conservation of energy is guaranteed.

The hydraulics library includes different storage elements like open vessels, spring-loaded tanks and other accumulators, pipes with laminar and turbulent flow characteristics as well as various valves. The inertia of the fluid flowing through the pipe is responsible for the inductive effect. In all three system categories (capacitance, resistance and inductance) the stored or dissipated energy is calculated. In an additional resistor element, the entropy is recalculated and is connected by a thermal connector. The temperature of this connector determines the viscosity of the fluid.

Two ideal pumps are modeled, one with a signal input for pressure difference and one with a signal input for the volume flow rate. In addition, the library contains sensors for pressure and volume flow.

### 2.2 Electrodynamics

The electrodynamics library based on the Modelica standard libraries with capacitor, resistor, diode and inductor. In addition, an isolated metal sphere is modeled for charge storage in experiments with high voltage.

The energy balance is calculated in all elements. This calculation is made for didactic purpose and for energy check in complex systems. A resistor with a thermal connector wherein the entropy is calculated enables the modeling of electro-thermal elements such as resistance heating or light bulb.

### 2.3 Mechanics

The mechanics library includes two parts: one for motion along a straight line and one for rotation around a fixed axis. Therefore, the associated quantities momentum and angular momentum can be treated as scalars. The similar statement is true for the associated potentials (velocity and the angular velocity). The tight separation of the balance equation for momentum or angular momentum from kinematics has some implication: mass and moment of inertia have only one connector and distance or angle can only be calculated in elements that describe the flow of momentum or angular momentum.

In addition to the linear systems some more elements, such as friction, air resistance or elastomer spring are modeled. An ordinary rope or string is a further element often used in physics instruction. It is modeled as a spring-damper-system with predetermined breaking point. As in hydraulics and electrodynamics some elements for momentum or angular momentum flow are provided with a heat connector. The produced entropy is calculated and the temperature has an influence on the constitutive laws of these elements.

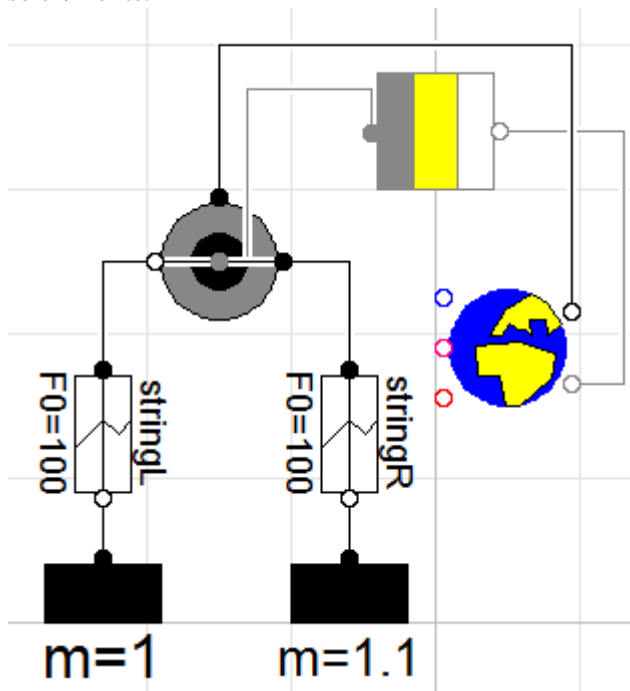


Figure 1: A model of an Atwood machine with two weights and bearing friction. The symbolic earth with five connectors for the flow of volume, momentum, angular momentum, electric charge and entropy stands for the surrounding.

The mass element has a momentum source which strength corresponds to the tangential component of the weight force. A further element contains the equation for the relativistic mass. In this element,

energy and momentum are connected with the help of the famous Einstein equation. The model of a simple rocket engine completes the model zoo, although this element belongs to the open systems, which are not included in this library. Figure 2 shows the velocity-time behavior of a rocket another system that can be easily modeled with our approach.

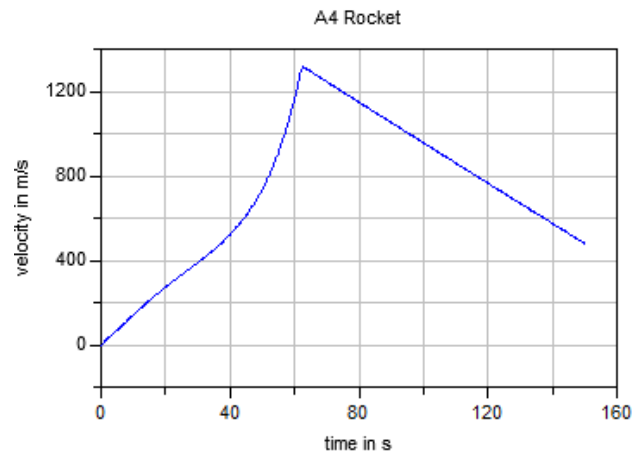


Figure 2: Velocity-time-diagram of a rocket ascending in an isentropic atmosphere with constant gravitational field

Translational and rotational mechanics are connected to each other by means of pulleys. The corresponding model has four connectors, three for translation and one for rotation. With two bodies, two strings, a pulley and a bearing friction, we can model Atwood's machine (Figure 1).

### 2.4 Thermodynamics

If we take entropy as the basic quantity of thermodynamics it's easier to write down the correct equations than if we take the energy as a conserved quantity [6]. However, for the equations themselves, it does not matter whether we start from the energy or from entropy as we have shown in the introduction. There are two different models which specify the transport of heat, the heat conduction and the ideal heat engine.

We describe homogeneous systems, which are heated at a constant pressure, with the state variable enthalpy  $H$ . Enthalpy is a special form of energy and a thermodynamic potential. Although entropy flow and temperature are calculated in the connector, this is no problem with respect to the balance equation for energy

$$TI_s = \frac{dH}{dt} \tag{10}$$

More generally, a homogeneous thermodynamic system can at least change entropy and volume. Therefore the system has temperature and pressure as two associated potentials. To discuss and model

such a system we have developed the Carnotor, a simple machine with a thermal and a hydraulic connector (Figure 3). Carnotor is a portmanteau composed of Carnot and Motor (German word for engine).

The Carnotor consists of a double-acting cylinder filled with the substance to be examined on one side of the piston and an ideal fluid on the other side. To each port we can add a pump, a closing-off or a big storage tank. With this equipment students can analyse all four basic processes of thermodynamics (Table 2).

The corresponding model calculates pressure and temperature from the change of volume and entropy. For the ideal Gas, the constitutive laws are as follows

$$S = S_0 + nR \ln\left(\frac{V}{V_0}\right) + \frac{f}{2} nR \ln\left(\frac{T}{T_0}\right); \quad (11)$$

$$pV = nRT \quad (12)$$

$f$  stands for the degrees of freedom of gas molecules,  $n$  for the amount of substance and  $R$  for the gas constant. Equation (12) is known as ideal gas law.

Table 2: The four basic processes in thermodynamics

process	heat port	hydraulic port	unchanged
isochoric	heat pump	closing-off	volume
isobar	heat pump	storage tank	pressure
isentropic	closing-off	hydraulic pump	entropy
isotherm	storage tank	hydraulic pump	temperature

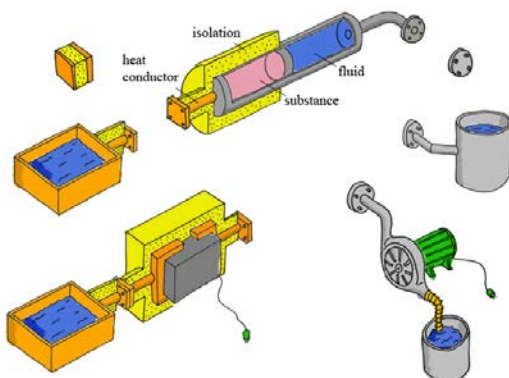


Figure 3: The Carnotor has two ports, one for heat and one for an ideal fluid. Both ports can be combined with a closing-off, a storage tank or a pump

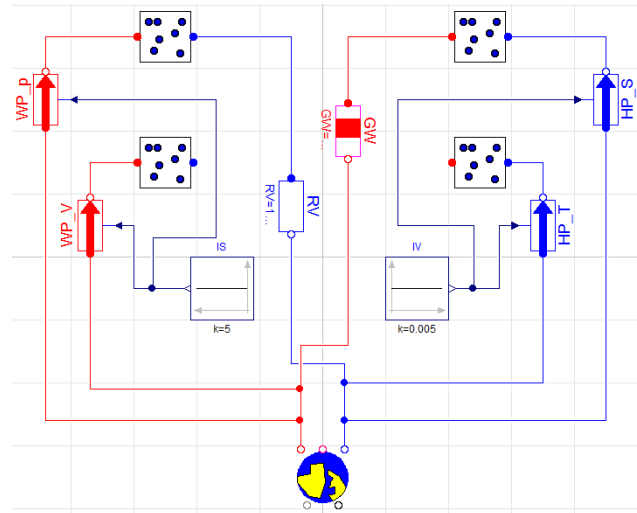


Figure 4: This system contains four models for ideal gas, two ideal heat pumps, two ideal hydraulic pumps, a heat flow and a volume flow element.

The Carnotor can be taken as the core element for a lot of thermodynamics engines. Figure 4 shows a model with which one can simulate all four basic processes simultaneously

### 3 Conclusions

Systems Physics provides a consistent, coherent and relevant structure of physics. A huge number of dynamical systems can be modeled with the same heuristic approach. The equation of balance for substance-like quantities like volume, mass, electric charge, momentum, angular momentum and entropy yields the backbone for such models. By adding the constitutive laws for accumulators and conductors we get the basic equations. In a third step we can add energy as a second substance-like quantity. The energy balance analysis is often useful but not necessary for simple systems. But energy conservation becomes an inevitable requirement in more complex systems like thermodynamic accumulators.

Systems Physics has been developed on the basis of the Karlsruher Physikkurs [7] and taught in different physics courses at Zurich University of Applied Sciences.

With the help of the *Systems Physics library* we hope that we can convince more and more teachers of the usefulness of this method. A countless number of dynamic models are waiting to be modeled with a System Dynamics or a Modelica tool.

On Youtube you can find some tutorials on specific topics of the *Systems Physics library* [8]

## References

- [1] <http://code.google.com/p/system-physics/>
- [2] Borer, T et al.(2010). Physik – Ein systemdynamischer Zugang für die Sekundarstufe II. hep, Bern.
- [3] Maurer, W (2002). Systemdynamik – Ein möglicher Pfad durch den Irrgarten von Fehlvorstellungen. PdN-PhiS. 7/51, Aulis, München.
- [4] Karnopp, D., Margolis, D., Rosenberg, R. (1990). System dynamics: a unified approach. Wiley, New York.
- [5] Maurer W. Physik und Systemwissenschaft in Aviatik. Proceedings of the 20th Symposium on Simulationstechnique ASIM 2009, Cottbus, Germany, ASIM September 23-25, 2009.
- [6] Fuchs H (2010). The dynamics of heat. Springer, New York.
- [7] 2005.Friedrich Herrmann et al. (2010): Der Karlsruher Physikkurs. Aulis, München.
- [8] <http://www.youtube.com/user/Systemdynamiker> (playlists *Modelica-Kurs* and *Modelica*).



# Implementation of the Omni Vehicle Dynamics on Modelica

Ivan Kosenko<sup>1</sup> Kirill Gerasimov<sup>2</sup>

<sup>1</sup>Moscow State Technical University of Radio Engineering, Electronics, and Automation  
Department of Engineering Mechanics  
Vernadsky Avenue 78, 119454, Moscow, Russia

<sup>2</sup>Lomonosov Moscow State University  
Department of Theoretical Mechanics and Mechatronics  
Leninskiye Gory 1, Main Building, GSP-1, 119991, Moscow, Russia

## Abstract

Omni wheel is defined as one having rollers along its rim. Respectively omni vehicle is one equipped by omni wheels. Several steps of development of dynamical model of the omni vehicle multibody system are implemented. Essential parameters of the model: (a) number of rollers per the wheel, and (b) angle of the roller axis inclination to the wheel plane, are introduced. Initially, dynamics of the free roller moving in a field of gravity and having a unilateral contact constraint with horizontal surface is modeled. The contact tracking using simplified and efficient algorithm turns out being possible. On the next stage the omni wheel model is developed and debugged. After that the whole vehicle model is assembled as a container class having arrays of objects as instantiated classes / models of omni wheels and joints. Dynamical properties of the resulting model are illustrated via numerical experiments.

*Keywords: omni wheel; contact tracking; unilateral constraint; contact detection; model of friction*

## 1 Introduction

Investigation of omni vehicle dynamical properties is sufficiently popular topic in frame of the multibody dynamics [1, 2, 3, 4]. The omni vehicle is one having omni wheels, wheels equipped by rollers along the rim. Simplified, idealized models having contacting rollers as an infinitely small discrete elements are known. Thus one has a resulting non-holonomic constraint being “uniformly distributed” over the wheel rim. As a result, paradoxically, the physical objects, omni wheels, describe approximately, in this situation, an idealized object, “simplified” infinitesimal model.

Our goal in this paper is to develop a technique for

building up a dynamical prototype for the “real” model of the omni vehicle explicitly involving dynamics of physical rollers. Here we rely upon the “simple” 3D multibody dynamics library classes utilized previously in several examples of the multibody systems dynamics [5]. Simultaneously this library enables us to create complex dynamical models including unilateral constraints of different nature.

Unlike to [2, 3] we emphasize here on the details of the unilateral constraint implementation paying special attention to contact switching when rollers changing.

## 2 Problem formulation

Upon describing the omni vehicle model construct note that the number of rollers per each wheel and the angle of inclination of the roller axis of symmetry to the wheel plane are both fundamental parameters of the vehicle dynamical model. For simplicity and presentation clarity we currently consider omni wheels being equipped by four rollers. Also, for simplicity rollers themselves have their axes of symmetry lying in the wheel plane, see Figure 1. These fundamental parameters are easily changed in a way simple enough. We assume also that the rollers are located on the omni wheel such that for wheel vertically aligned a projection of the curve of contact consists of segments in the sequence, each segment corresponding to the contact of individual roller. These segments are connected in a way such that the normal relative velocity at contact is equal to zero at the switching point of rollers. This means that the normal impact is always absent. Discontinuities of the tangent relative sliding velocity are absent for zero angle of inclination. But the tangent force of friction may have discontinuity of the first kind in the worst case of angle of inclination if the

roller symmetry axis to the wheel plane is non-zero. Thus, the wheel linear and angular velocities will be continuous at an instant when roller switching contact. Similar statement takes place for rollers, as well. Then tangent impacts are also absent.

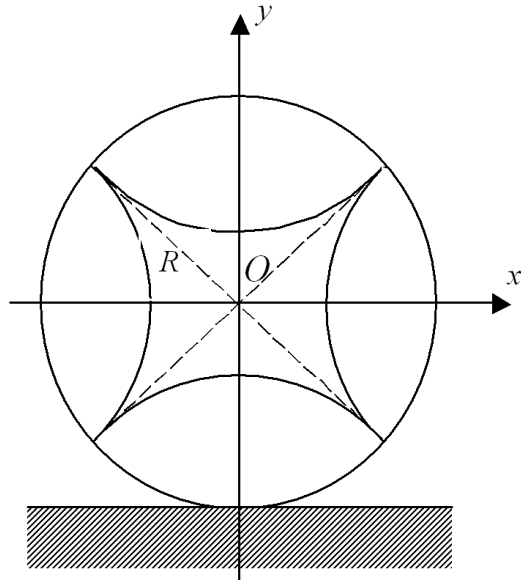


Figure 1: The omni wheel vertically aligned.

Note, in addition, that the curve of contacting points forms the  $xy$ -projection onto the wheel plane, having a shape of the circle of radius  $R$ , see Figure 1. Thus for translational and rotational motion we have continuity as well. Resuming we are able to conclude that the regularity of motion is conserved as roller switching at contact. At least on the level of integrity of the omni wheel. Recall, that all the description above takes place for vertical alignment of the omni wheel.

On the next level of assembling process, several omni wheels are interconnected with the moving platform of the vehicle, see Figure 2, using joint constraint as a class from the previous stage. In our case, number of wheels may be three or more. They can form different configurations on the platform body. We analyse an example with three wheels forming an equilateral triangle in the plane of the platform, see Figure 2, parallel to the coordinate horizontal plane  $zx$ . Axis  $y$  here is assumed vertical.

### 3 The roller dynamics model

Firstly, we presume that the roller is axisymmetric spindle-shaped rigid body having outer surface defined in body frame of reference  $Oxyz$ , see Figure 3, by

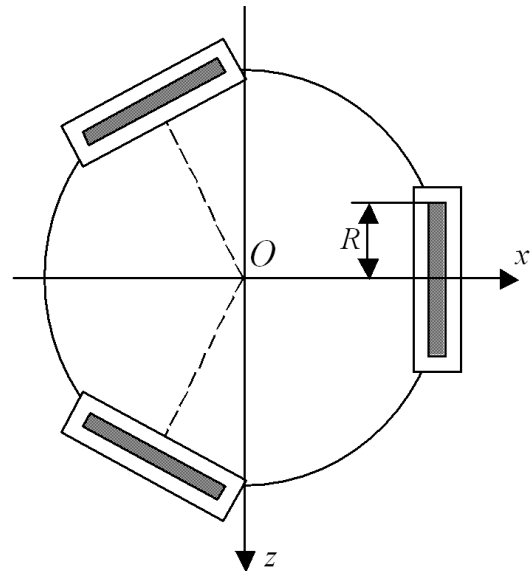


Figure 2: The three wheeled vehicle. Top view.

equation

$$x^2 + \left( \sqrt{y^2 + z^2} + R_1 \right)^2 = R, \quad (1)$$

where  $R$  is the omni wheel radius,  $R_1 = R \cos \alpha$  is the distance from the roller center to the wheel center,  $\alpha = \pi/n$  is the half of the roller opening angle of visibility from the wheel center,  $n$  is number of rollers per wheel.

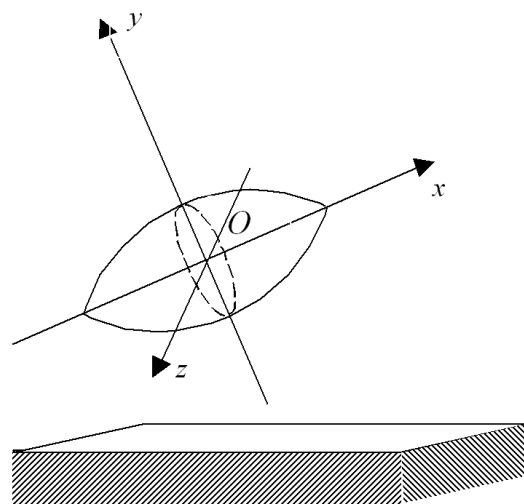


Figure 3: The roller over horizontal surface. Lateral view.

Dynamics of the roller translatory–rotary motion is implemented using equations of Newton – Euler as was shown in [6]. And rotational motion was modeled by the quaternion algebra [7].



Algorithm for contact tracking plays an important role for the correct and efficient work of the computer model of the contacting process of the roller and the horizontal surface. For modeling and simulation of the rigid body dynamics with unilateral constraint we apply the technology described in [8]. So we could have used in the object of contact a system of well known algebraic or implicit differential–algebraic equations. However, these equations degenerate at the roller tips defined by equations  $x = \pm R \sin \alpha$  in the roller coordinates, see Figure 1. Usually, such a degeneration causes an abnormal completion of the simulation process.

In our case of the spindle-shaped roller over the horizontal surface, arranging the contact tracking procedure turns out being sufficiently simple. So one can point out explicit procedure for computing the nearest point  $P_B$  of the roller to the horizontal surface, see Figure 4. This surface has its own nearest point  $P_A$  at contact. Evidently the point  $P_A$  is a vertical projection of the point  $P_B$  of the roller.

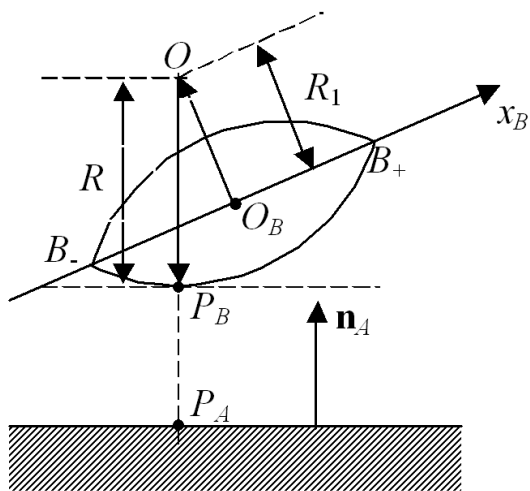


Figure 4: Contact tracking scheme.

Denote by  $\mathbf{i} = (1, 0, 0)^T$  the unit vector along the axis  $O_B x_B$  of the roller connected coordinate system from Figure 4. This vector is resolved with respect to (w. r. t.) the system  $O_B x_B y_B z_B$ . Let  $T_B$  be the rotational matrix of the roller w. r. t. the inertial frame of reference. The latter frame, in our case, coincides with the fixed horizontal surface coordinate system  $O_A x_A y_A z_A$ . Also, let  $\mathbf{r}_B$  be the roller current mass center radius vector w. r. t. the inertial system, and  $\mathbf{n}_A = (0, 1, 0)^T$  be the ascending vertical unit vector. Simultaneously  $\mathbf{n}_A$  is the normal vector to the horizontal plane.

Conventionally, we denote the plane as body A, and

roller as body B. Let  $\mathbf{d}$  be the horizontal unit vector defined by equation

$$\mathbf{d} = \frac{T_B \mathbf{i}_B \times \mathbf{n}_A}{|T_B \mathbf{i}_B \times \mathbf{n}_A|}.$$

Therefore, the directed segment  $\overrightarrow{O_B \mathcal{O}}$  must have a length  $R_1$  and be defined by formula

$$\overrightarrow{O_B \mathcal{O}} = R_1 \mathbf{d} \times T_B \mathbf{i}_B.$$

Here,  $O$  is the curvature center for the circle of the roller vertical section, see Figure 4. This segment is located simultaneously in the vertical plane and in the wheel plane. Thus from Figure 4 we see that the lowest point  $P_B$  of the roller outer surface is defined by equation

$$\mathbf{r}_{P_B} = \mathbf{r}_B + R_1 \mathbf{d} \times T_B \mathbf{i}_B - R \mathbf{n}_A \quad (2)$$

since the  $P_B$  lies on the same vertical with the point  $O$  and on the circle mentioned above. To compute position of the point  $P_A$  one has to put

$$\mathbf{r}_{P_A} = (x_{P_B}, 0, z_{P_B})^T. \quad (3)$$

All the procedure above is valid only if the vector  $T_B \mathbf{i}_B$  is inclined to the horizontal plane within an angle  $\pm \alpha$ . Otherwise one has to put  $P_B = B_-$  where  $B_-$  is the “left”, see Figure 4, tip of the roller for angle of the vector  $T_B \mathbf{i}_B$  inclination greater than the value  $\alpha$ . If this angle is less than  $-\alpha$  then one has to guess  $P_B = B_+$  where  $B_+$  is the “right” tip of the roller.

Finally, one can write down a contacting condition between roller and horizontal surface in the form

$$|T_B \mathbf{i}_B \cdot \mathbf{n}_A| \leq \sin \alpha. \quad (4)$$

This condition, however, is satisfied simultaneously for the lowest, being in contact, roller, and the highest one. To reject the latter case one can add to condition (4) yet another one

$$y_B < R \quad (5)$$

where  $y_B$  is the altitude of the roller mass center w. r. t. inertial frame of reference.

So a conjunction of conditions (4) and (5) is equivalent to the case of contacting. Otherwise condition of normal reaction being zero should take place. Indeed, according to Signorini’s law a following alternative is implemented for each individual roller: (a) contact takes place – relative normal velocity at contact should be zero; (b) contact is absent – normal reaction (and tangent too) of unilateral constraint should be zero.

Condition (a) has several alternative possibilities of implementation. Firstly, from the geometric viewpoint a presence of contact is equivalent to the scalar condition

$$y_{P_B} = 0. \quad (6)$$

Its absence is equivalent also to the scalar condition

$$F_n = 0$$

where  $F_n$  is the normal component of a reaction force acting on the roller at the point  $P_B$ .

Computational experience show that equation of contact in the form (6) usually causes an abnormal termination of the simulation process for the dynamical model of the roller. One has similar result if we use equation

$$v_n = 0$$

as an implementation of condition (a). Here  $v_n$  is the normal component of the relative velocity at contact point. And only equation of the form

$$\dot{v}_n = 0$$

leads to the required result: object of contact works properly during the simulation process. One has to recall here that all the implementation of the contacting process has the “rigid” point-contact model.

For each roller of the omni vehicle model when contacting the friction model being used is “turned on”. In our model being developed the “simple” law of the Amontons – Coulomb dry friction is applied. Actually we use known piecewise approximation [8] to exact dry friction instead. This approximation has high accuracy over long time intervals [9]. In general, implementation of unilateral constraint model is based on the results outlined in [8].

If the angle of inclination for the roller axis of symmetry to the wheel plane has non-zero value then some of the above relations ought to be slightly corrected. In this case, rollers become distorted along the wheel rim. Given the position  $\mathbf{r}_O \in \mathbf{R}^3$  of the wheel center, point  $O$ , see Figure 4, firstly, we have to build up an auxiliary base consisting of unit vectors:

$$\mathbf{i}' = T_B \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{j}' = \frac{\mathbf{r}_O - \mathbf{r}_{O_B}}{|\mathbf{r}_O - \mathbf{r}_{O_B}|}, \quad \mathbf{k}' = \mathbf{i}' \times \mathbf{j}'.$$

After that a matrix of coordinates change has the form  $T' = (\mathbf{i}'\mathbf{j}'\mathbf{k}')$  where  $\mathbf{i}', \mathbf{j}', \mathbf{k}'$  are assumed as vector columns. This matrix defines transformation from inertial frame of reference connected with the fixed

body  $A$  to the frame defined by the vector base  $B' = \{\mathbf{i}', \mathbf{j}', \mathbf{k}'\}$  introduced above in the following way

$$\begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} = T' \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}.$$

To reduce an analysis to the case of  $\beta = 0$  already considered above we have to rotate the base  $B'$  about  $\mathbf{j}'$  by the angle  $-\beta$  such that after the rotation a new base  $B = \{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$  should be aligned with the wheel plane containing the unit vectors  $\mathbf{i}, \mathbf{j}$ . The rotation mentioned has the matrix

$$S = \begin{pmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{pmatrix}$$

in the base  $B'$ . Then in the base of the indicated body  $A$  the rotation of the unit vector  $\mathbf{i}'$  can be represented as follows  $\mathbf{i} = T'S(1, 0, 0)^T$ . Suppose also  $\mathbf{j} = \mathbf{j}', \mathbf{k} = \mathbf{i} \times \mathbf{j}$ . Evidently  $\mathbf{k} = \mathbf{d}$  where  $\mathbf{d}$  is the unit vector given above.

Thus based on the formula (2) and taking into account Figure 4 we can conclude that for the case of  $\beta \neq 0$  the following result takes place

$$\mathbf{r}_{P_B} = \mathbf{r}_B + R_1 \mathbf{j} - R \mathbf{n}_A - \frac{R_1 \tan \beta \sin \gamma}{\sqrt{1 - \sin^2 \gamma}} \mathbf{j} \times \mathbf{i}, \quad (7)$$

where the angle  $\gamma$  satisfies the equation

$$\sin \gamma = \mathbf{i} \cdot \mathbf{n}_A.$$

## 4 Assembling vehicle model

An assembling process of the omni vehicle prototype is implemented in two steps: (a) assembling the omni wheel consisting of the wheel itself and a set of rollers attached to the wheel; (b) assembling the vehicle by instantiating objects of the omni wheel class from stage (a) into the container class of the vehicle prototype.

To connect rollers, rather objects of the roller class, and the wheel we use model of the joint constraint previously developed and described in [5]. It is simply revolute class with free relative rotation about its axis. Codes of all the classes / models for the prototype are implemented as Modelica classes library. See visual model of the omni wheel in Figure 5. Here, in our example we selected for simplicity and certainty  $n = 4$ .

The model of main interest is one of the whole vehicle which is “assembled” on the second stage of the assembling process. Connecting devices were also implemented as objects of the same joint class from stage

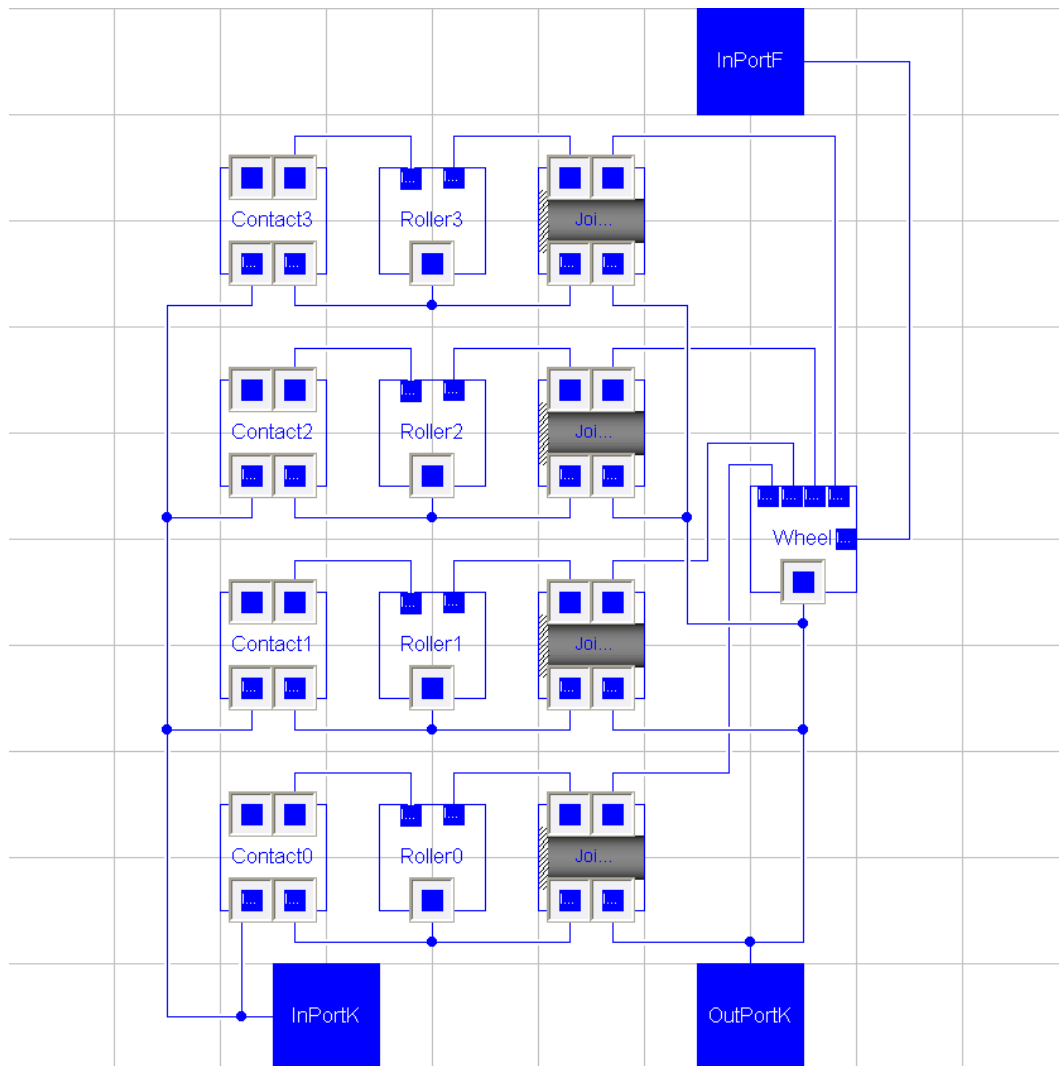


Figure 5: The omni wheel visual model.

(a). These joints connect the vehicle body and each of wheels. All joints above allow relative rotation without any resistance and lock sliding along the joint axis. See visual model of the vehicle in Figure 6. Here, for presentability, objects are shown as scalar elements. Actually, one has to instantiate corresponding arrays of objects of classes “Roller” and “OmniWheel” for arbitrary  $n$  and arbitrary number of wheels in the vehicle.

Recall that before the DAE index reduction process implemented in Dymola the whole vehicle model consists of: (a) one rigid body of the vehicle platform; plus (b) three rigid bodies of the vehicle wheels; plus (c) twelve rigid bodies of rollers located on the wheels. According, for instance, to [5] for each object of rigid bodies we implement six Newton’s ODEs for the mass center motion plus seven Euler’s ODEs for rotational motion about the mass center. For the latter case we have four Euler’s kinematical equations for the rigid

body quaternion plus three Euler’s dynamical equations for the rigid body angular velocity. Totally, the whole vehicle model includes system of ODEs of order  $16 \cdot 13 = 208$ . Besides, constraint objects are able to generate additional differential equations.

Wheels being assembled into the vehicle will keep the vertical alignment unavoidably. For this reason the simplified contact tracking algorithm described above works properly.

Computer experiments were performed for different numbers of rollers per wheel and using several frictional models at contact between roller and the horizontal surface. Corresponding results were compared. For instance an evolution of the contact process for one wheel of the three wheeled vehicle is shown in Figure 7. Paying attention to the Figure legend we are seeing variables with suffixes “.h” and respectively curves of four colours. This variables represent so called mutual approaches for contacting bodies. Their values are simply distances between rollers of

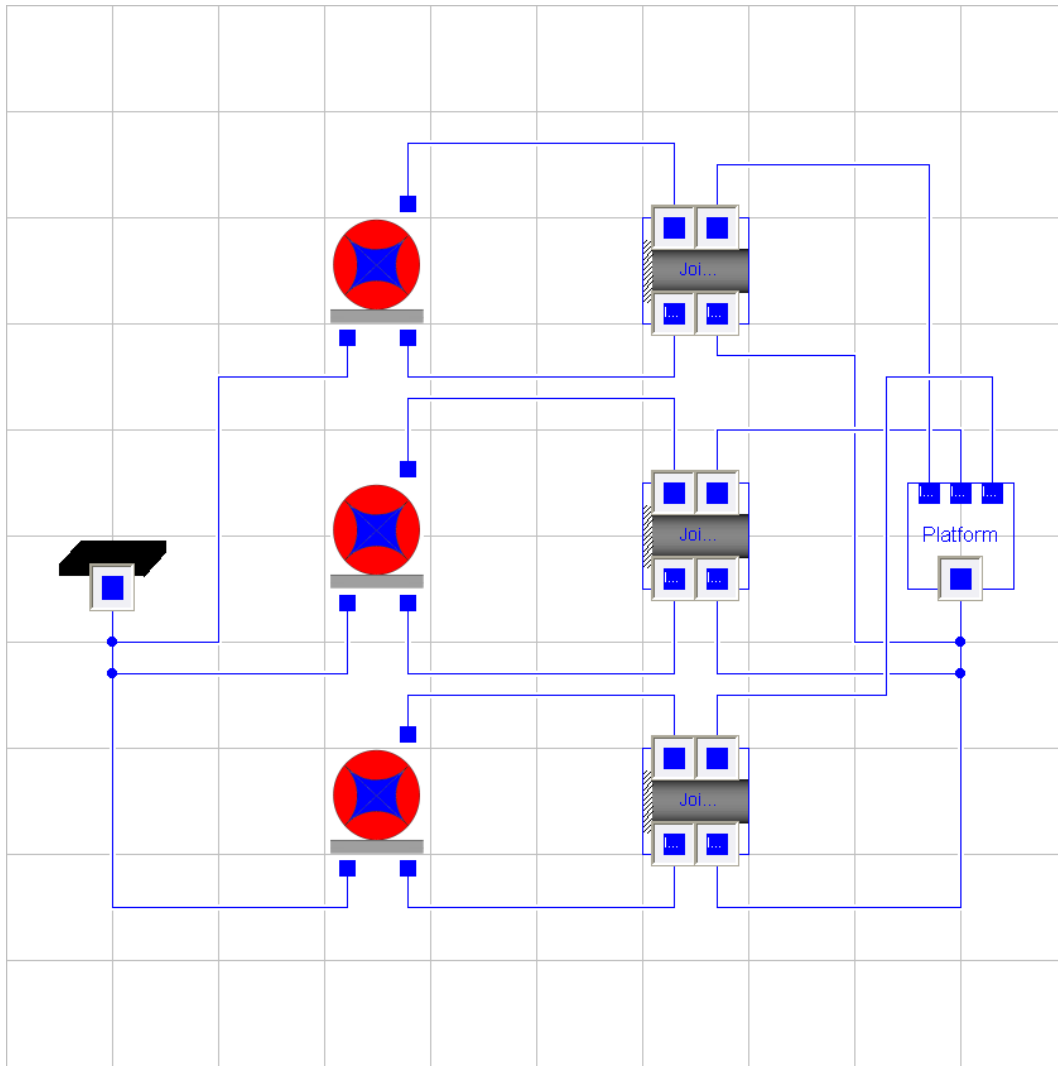


Figure 6: The omni vehicle visual model.

the wheel and the horizontal surface of rolling. These curves correspond to rollers being in different phases of wheel rotation: before contact, at contact, after the contact. See an instance of the roller change being zoomed in Figure 7. For implementing such a switching it is sufficiently simple to use if-clause thus alternating states of the contact existence / non-existence. Corresponding fragment of Modelica code may have the following representation

```

...
if noEvent(abs((T_B*i)*nA) < cos_of_max and
             h < R) then
  Drelvn = 0;
  Forcet = -fric*relvt*(if
                       noEvent(relvtsqrt <= delta)
                       then 1/delta
                       else 1/relvtsqrt)*Forcen + mu*nA;
else
  Forcen = 0;
  Forcet = zeros(3);
end if;

```

```

Drelvn = der(relvn);
...

```

The first if-operator here is responsible for the unilateral constraint detection. Its condition is equivalent to conjunction of conditions (4) and (5). Drelvn is the variable being equal to the derivative of the relative normal velocity at contact. So we have an alternative: (a) Drelvn = 0 means the contact existence or (b) Forcen = 0 means contact absence or, equivalently, zero-valued force of reaction. Vector variable Forcet simulates tangent force of friction, being computed here using piece-wise linear approximation of dry friction.

Simultaneously, one can also observe the unilateral constraint accuracy being kept by the model at contacting, see Figure 8. In this Figure we can observe how a numeric error of the unilateral constraint feasibility slowly diverges, mutual approach . . . h gradually

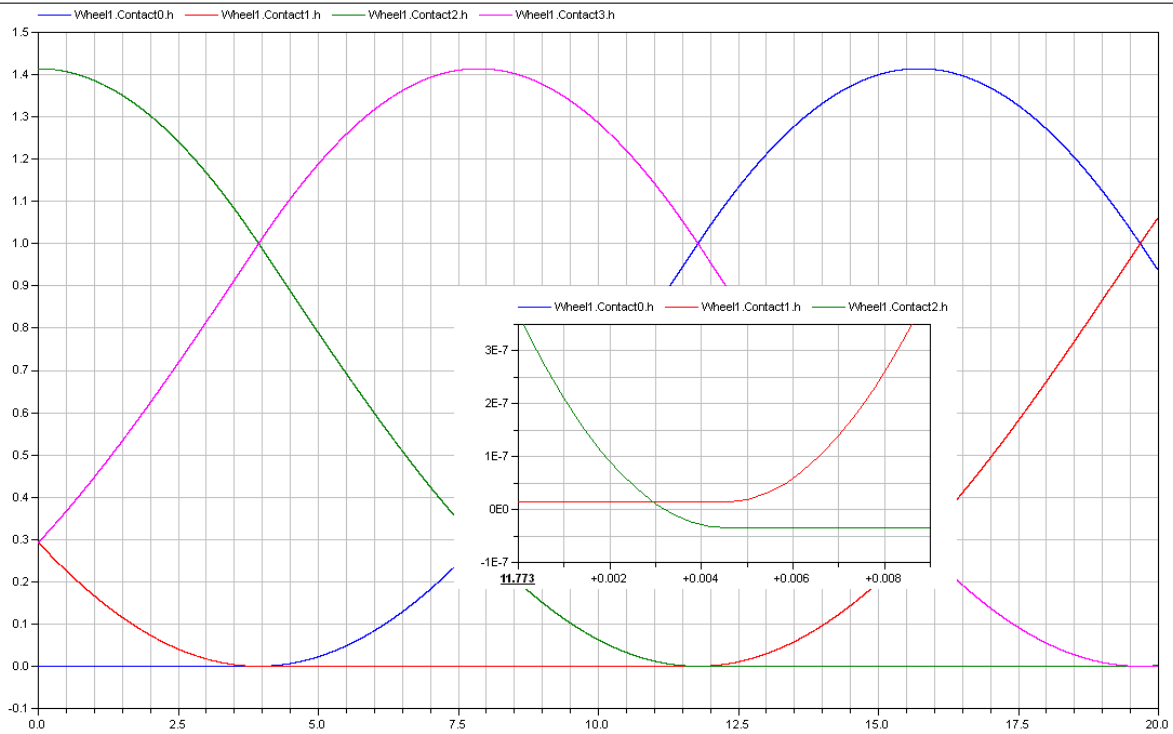


Figure 7: Process of rollers contact replacement

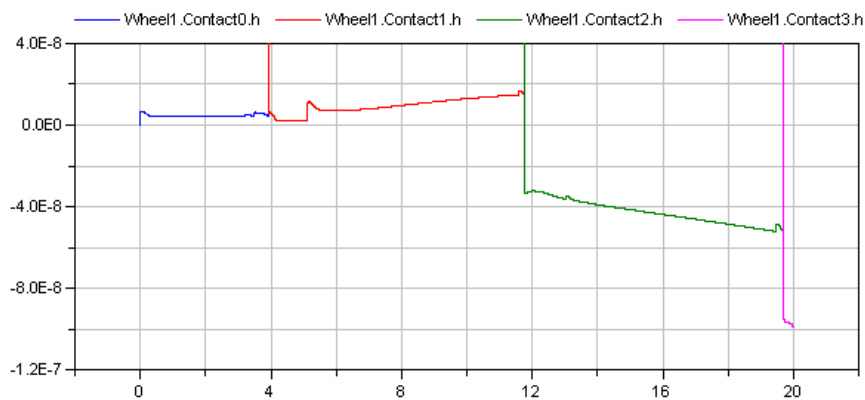


Figure 8: Accuracy of the unilateral constraint.

grows, for each successive roller in contact. Meanwhile, an absolute value of error stays near negligible value of  $10^{-7}$  meters. Change of the curve colour corresponds to change of the contacting roller.

## 5 Conclusions

As a summary of main results obtained in the course of the omni vehicle model development we can highlight the following issues:

- There exist a possibility for smooth impactless switching between rollers at contact upon rolling of omni wheel;
- Efficient and simplified contact tracking algorithm was implemented;

- Dynamics of vehicle was investigated for different number of rollers per wheel;
- Influence of friction model on dynamics of the omni vehicle was analyzed.

This work was performed with partial support of RFBR, projects 11-01-00354-a, 12-01-00536-a, 12-08-00637-a.

## References

[1] Campion, G.; Bastin, G.; d'Andréa-Novel, B.: Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots. IEEE Transactions on Robotics and Automation, Vol. 12, No. 1, pp. 47–62, 1996.

- [2] Kálmán V.: Controlled Braking for Omnidirectional Wheels. *International Journal of Control Science and Engineering*, Vol. 3, No. 2, pp. 48–57, 2013.
- [3] Tobolár, J.; Herrmann, F.; Bünte T.: Object-oriented modelling and control of vehicles with omni-directional wheels. *Computational Mechanics 2009*. Hrad Nectiny, Czech Republic, November 9–11, 2009.
- [4] Zobova, A. A.; Tatarinov, Ya. V.: The Dynamics of an Omni-Mobile Vehicle. *Journal of Applied Mathematics and Mechanics*, Vol. 73, Iss. 1, pp. 8–15, 2009.
- [5] Kosenko I. I., Loginova M. S., Obraztsov Ya. P., Stavrovskaya M. S., *Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation*. In: *Proceedings of the 5th International Modelica Conference*, arsenal research, Vienna, Austria, September 4–5, 2006, pp. 213–223.
- [6] Kossenko I. I., Stavrovskaja M. S., *How One Can Simulate Dynamics of Rolling Bodies via Dymola: Approach to Model Multibody System Dynamics Using Modelica*. In: *Proceedings of the 3rd International Modelica Conference*, Linkopings universitet, Linköping, Sweden, November 3–4, 2003, pp. 299–309.
- [7] Kosenko, I. I., *Integration of the Equations of the Rotational Motion of a Rigid Body in the Quaternion Algebra. The Euler Case*. *Journal of Applied Mathematics and Mechanics*, 1998, Vol. 62, Iss. 2, pp. 193–200.
- [8] Kossenko I. I., *Implementation of Unilateral Multibody Dynamics on Modelica*. In: *Proceedings of the 4th International Modelica Conference*, Hamburg university of technology, Hamburg–Harburg, Germany, March 7–8, 2005, pp. 13–23.
- [9] Novozhilov, I. V., *Fractional Analysis : Methods of Motion Decomposition*. Boston: Birkhauser, 1997.

# Control and Characteristic Map Generation of Permanent Magnet Synchronous Machines and Induction Machines with Squirrel Cage

Marco Kessler  
Vorarlberg Univ. of Appl. Sc.  
Austria  
[marco.kessler@modelon.com](mailto:marco.kessler@modelon.com)

Markus Andres  
Modelon GmbH  
Germany  
[markus.andres@modelon.com](mailto:markus.andres@modelon.com)

Thomas Schmitt  
Modelon GmbH  
Germany  
[thomas.schmitt@modelon.com](mailto:thomas.schmitt@modelon.com)

## Abstract

Due to different requirements for simulation during the design process of an electric drive system, investigations were carried out to supply models of different stages with consistent sets of parameters. Therefore physical models of the Modelica Standard Library were equipped with state-of-the-art control strategies to operate in realistic conditions. In the presented case the losses computed by the physical model were stored in characteristic tables to speed up simulation in cases where dynamics are of minor interest. These models can then be used for energetic, thermal and life-time analysis with a consistent set of parameters generated from their physical counterparts.

*Keywords: PMSM, induction machine, characteristic maps, field oriented control, MTPA*

## 1 Introduction

The continuous progress in the electrification of the powertrain in the automotive industry requires numeric based computer simulations to handle the growing complexity. During development various types of simulations are needed, some of which are summarized in Table 1. Simulations of category 1 focus on brief events and need very detailed models for reliable predictions. In category 2 a tradeoff between accurate and fast models must be found, whereas category 3 and 4 especially require models which are optimized on computation time due to the long simulation runs.

The machine models of the Modelica Standard Library (MSL), which are described in detail in [3] and [1], are based on physical equations using space phasor theory. They consider leakages of the magnetic field with stray inductances and include basic models for the copper, stray load, core and friction losses.

Category	Objectives	Simulated time
1	switching operations nonlin. in the drivetrain	0.01 s – 1 s
2	dynamic events short-term performance controller behavior	0.1 s – 100s
3	energy consumption thermal behavior performance analysis	10s – $1 \times 10^4$ s
4	aging simulation life-time of components	$1 \times 10^6$ s – $1 \times 10^9$ s

Table 1: Categories for simulation

Hence, they are well suited for simulations of category 2. However, for their application in the electric powertrain control is required, which is offered in the commercial Smart Electric Drives Library (SED), but not in the MSL. For long simulation runs the SED Library also provides quasi stationary machine models [6], but these models only consider ohmic losses. Therefore, this paper describes the implementation of advanced control for the MSL machine models of the permanent magnet synchronous machine (PMSM) and the induction machine, to use them for the creation of characteristic maps for lookup table based map models. These map models include all losses which are modeled in the MSL machines in tables and allow significant improvements of the required computation times compared to the physical models. This advantage basically results from the negligence of dynamics caused by control and machine physics.

## 2 Control of the PMSM

The principle for field oriented control of the PMSM is to achieve a control concept known from the elec-

Name	Unit	Description
$i_d$	A	Direct current of PMSM
$i_q$	A	Quadrature current of PMSM
$i_{sx}$	A	Current for field generation in IM
$i_{sy}$	A	Current for torque generation in IM
$\omega_{nom}$	rad/s	Nominal angular velocity
$U_s$	V	Norm of stator voltage
$U_{sMax}$	V	Norm of max. allowed stator voltage
$\tau_{electrical}$	Nm	Electrical torque inside machine

Table 2: Variables used in the PMSMs control algorithm

trically excited DC machine, where the magnetic flux and the torque are controlled separately. The important variables in this section are shown in Table 2.

### 2.1 The Control Scheme

As this paper focuses on the computation of losses we will not discuss the control theory in detail. Still it is important to notice that the control scheme has major influence on the generated losses. Therefore a number of effects extending the basic concept of field-oriented control have been implemented in the machine control. These are

- Voltage limitation
- Current limitation
- Field weakening<sup>1</sup>
- Maximum torque per ampere

Although these effects influence losses, it is not in the scope of this paper to review how losses are changed based on different implementation of the control algorithm. An overview of the applied scheme is presented in Figure 1.

### 2.2 Simulation Results

Figure 2 depicts the obtained simulation results where a speed step of  $2 \cdot \omega_{nom}$  was applied to a permanent magnet synchronous machine. One can see in Figure 2 b) how  $U_s$  increases proportional with the speed until  $U_{sMax}$  is reached. Then the field weakening controller starts to decrease  $i_d$  and the speed continues to rise at an expense of a decreasing the inner torque  $\tau_{electrical}$ . In Figure 2 c)  $i_q$  is plotted over  $i_d$ , where the current vector moves along the maximum

<sup>1</sup>The algorithm used in this paper is based on a comparison of  $U_s$  and  $U_{sMax}$  and is based on theory presented in [4]

torque per ampere (MTPA) trajectory and then follows the current limit circle.

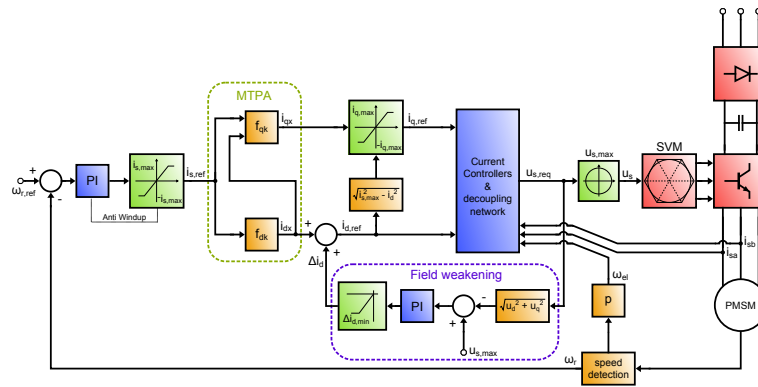
## 3 Induction machine

The control of the induction machine follows similar principles as it was presented for the PMSM. The stator voltage equation is used to create a decoupling network and to parameterize the current controllers. One difference is, that in general the rotor flux oriented reference frame is used to obtain separate control of the flux and the torque, which requires an estimator for the non-measurable rotor flux. The other difference is, that the flux must be generated, which is performed with the current component  $i_{sx}$ , whereas the  $i_{sy}$  is used for the torque generation. Differing from the PMSM the coordinate system is based on the  $x$ -axis that is aligned with the direction of the flux, whereas the  $y$ -axis is orthogonal to that and therefore responsible for generating the torque. They are named differently from the PMSMs coordinate system as the flux direction is moving at different speed from the rotors angular velocity. For more detailed informations it is referred to [5, Ch. 2 and 4.1.1] and [4].

In Figure 3 the control scheme of the induction machine is shown, which features voltage limitation, a field weakening controller and a controller for maximum torque in the upper field weakening region, which limits the output of the speed controller. For both machines, the PMSSM and the induction machine a torque controller was implemented additionally to the presented speed controller.

From the simulation results, illustrated in Figure 4, one can see how the rotor flux is generated with  $i_d$  in the first 0.5s. Then the speed increases until the voltage limit is reached, which requires to reduce the magnetizing current shown at the bottom right, which is used to create the rotor flux. As shown in the simulation results in Figures 2 and 4, a reliable control algorithm for both machine types was implemented. Thanks to the field weakening controllers and the output limitations it is ensured that the voltage and current limitations are not exceeded. Hence, the controlled machine models can be used in batch simulations in order to compute the characteristic map for a given set of machine parameters as well as stator current and stator voltage limitations.





Source: Based on [2]

Figure 1: Extended control scheme for the PMSM with MTPA and field weakening

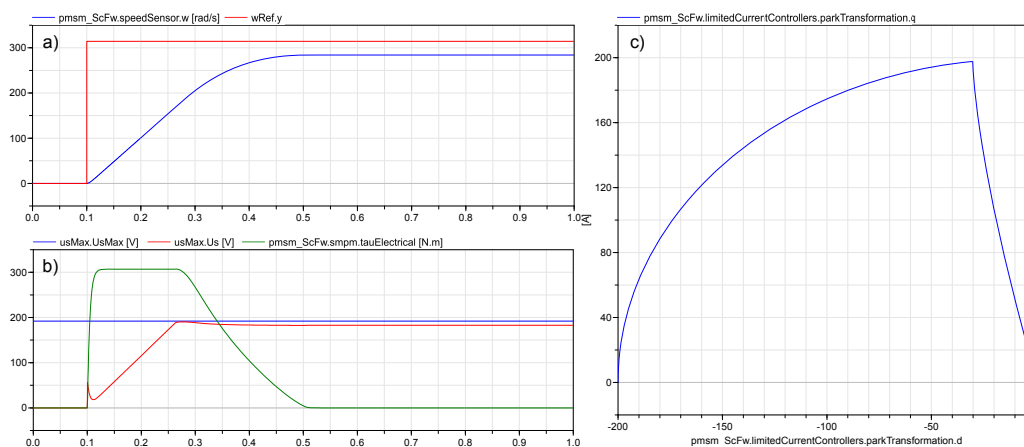


Figure 2: Simulation results when a speed step is applied to a speed controlled PMSM with field weakening (*pmsm\_ScFw*). (a) rotor speed rad reference and actual value; (b) voltage limit, mesured voltage and electric torque; (c)  $i_q$  over  $i_d$

### 4 Characteristic Maps

If a driving cycle is simulated for a hybrid or an electric car the energy consumption of different parts and the resulting attainable range are usually most important, whereas quantities like the magnetic flux or the voltage drops at the stray inductances are of minor interest. Therefore it is not required to have detailed physical models and the use of averaged models is sufficient. For electric machines such models can be composed of characteristic maps, in which the losses at various operating points are stored.

Characteristic maps of machines can also be used if the losses of a real machine are measured in different operating points. Then a convenient way to have an accurate model of this machine is to use the measurement results in the simulation within lookup tables. However, in an early design stage the real machine might not be available. Therefore in this paper

a method is presented which allows to calculate characteristic maps from simulations which are performed with physical models of a machine. In the paper the machine models from the Modelica Standard Library are used, but they can be replaced with more or less accurate models. This possibility is a major advantage compared to other methods e.g. averaged models that do simplifications to the models to make it compute faster. This way the modeling detail can be increased to a very high level only influencing the map generation time. As this has to be carried out once only for every new machine it is of minor importance compared to the many times the map-based model is simulated.

In the following sections it is shown how the maps can be generated and lookup table based models are presented which utilize those.



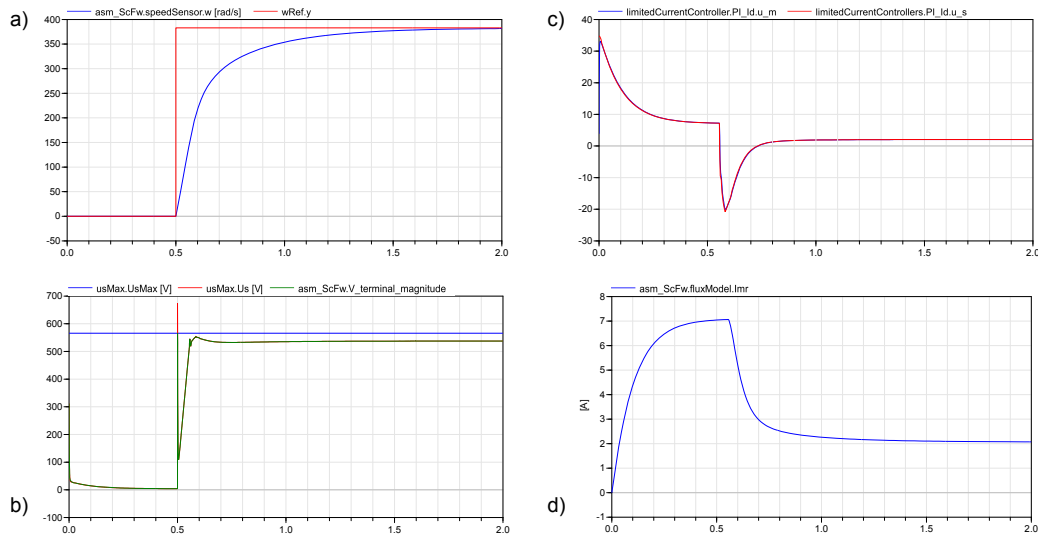


Figure 4: Simulation results of the induction machine: a) Reference and actual angular velocity; b) voltage limit, unlimited and limited voltage; c) current component  $i_d$ ; d) estimated magnetizing current

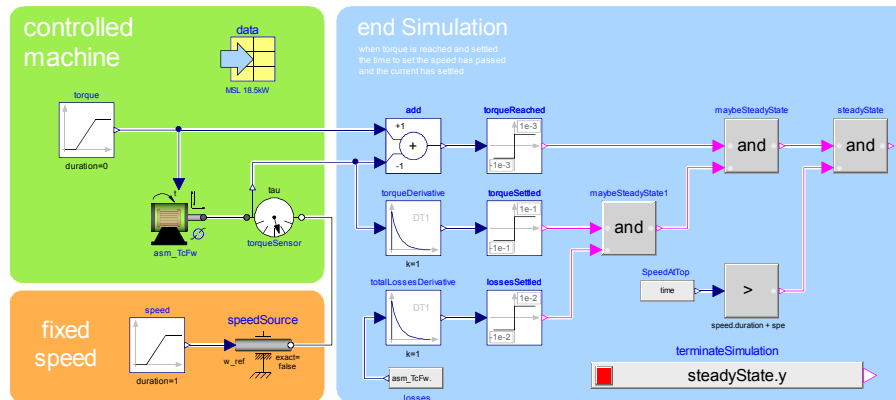


Figure 5: Model for loss calculation of a torque controlled induction machine

must specify the resolution of the map, which is used to determine which operating points will be simulated.

The calculated losses  $P$  of the simulated operating points are stored in matrices, with the first column containing the torque and the first row the angular velocity, resulting in

$$\begin{bmatrix}
 0 & \omega_{start} & \omega_2 & \dots & \omega_{stop} \\
 0 & P_{1,1} & P_{1,2} & \dots & P_{1,y} \\
 \tau_2 & P_{2,1} & P_{2,2} & \dots & P_{2,y} \\
 \tau_3 & P_{3,1} & P_{3,2} & \dots & P_{3,y} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 \tau_{stop} & P_{x,1} & P_{x,2} & \dots & P_{x,y}
 \end{bmatrix} \cdot \quad (1)$$

The ascending values in the first row and column are required for the usage in the lookup tables *Modelica.Blocks.Tables.CombiTable2D*. The range for the angular velocity is given by the lowest and the highest entry

of the user. The torque is simulated from zero up to the highest torque value which was entered. The area in between is discretized according to the user chosen resolution.

In Figure 7 an exemplary maximum torque curve and the performed discretization are illustrated. Due to arbitrary user inputs for the torque curve and the discretization, it is not possible to always simulate exactly up to the maximum torque curve. Hence, to cover the desired region it is required to perform one simulation above the curve. This is also illustrated in Figure 7, where simulated points are marked with  $\times$ , whereas for non-simulated points circles are used.

Since the whole matrix presented in Equation 1 has to be filled with values, the points marked with circles in Figure 7 have to be guessed somehow. To do so, the function *Modelica.Math.Vectors.interpolate* is used to

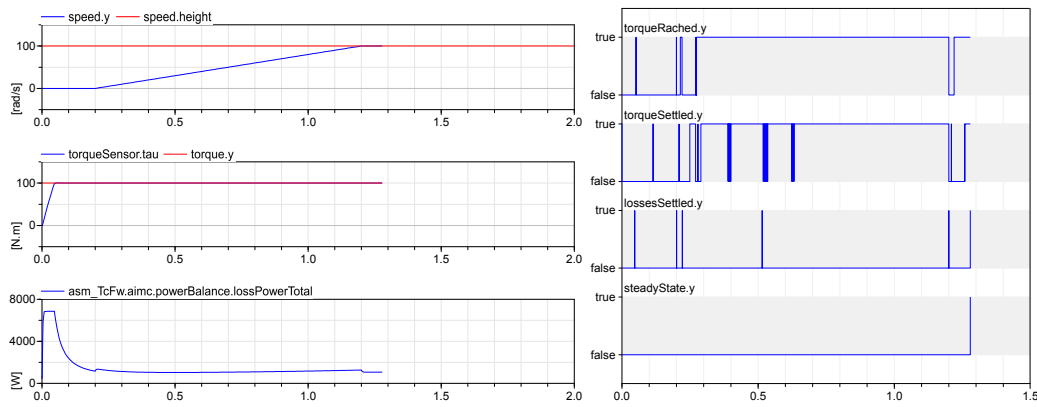


Figure 6: Simulation results of *TorqueControlledPhys2Map\_ASM*

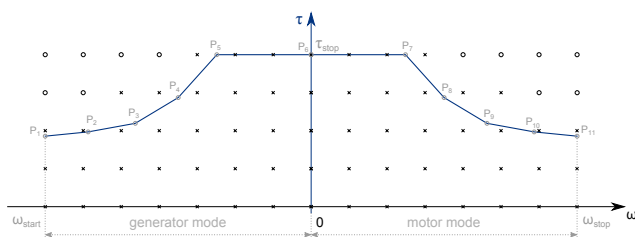


Figure 7: Exemplary maximum torque line and the performed discretization for a map of  $5 \times 15$

perform linear extrapolation from simulated values, which it does although the name would not indicate that. The extrapolated values for not simulated operating points in Equation 1 are calculated from the two rows above. This and the linear extrapolation might not be suited very well to calculate the correct values, but since it is used for the area above the maximum torque curve, the machine will hardly enter this region and it is still a much better approximation compared to leaving the losses on zero.

## 4.2 Model to utilize Characteristic Maps

To use the calculated maps in simulations, the model *TorqueControlledCharacteristicMap* is used, which is illustrated in Figure 8. It features two electrical DC connectors to the left, a mechanical connector to the right and a real input for the desired torque at the top. Hence, it is meant to replace a controlled electric machine with inverter. Via a parameter the .mat file containing the losses is specified.

The torque reference given with the real input *desiredTorque* is limited by *limitTorque* according to the maximum torque line, which the user has defined during map generation. Thereafter the first or-

der element *firstOrder* introduces a delay between desired and obtained torque<sup>2</sup>. The first order element is connected to the torque source *mechanicalTorque*, which accelerates *rotorInertia*. Parameters are provided in the model, which allow the user to deactivate *limitTorque* and *firstOrder*. Then the gain blocks to *noTorqueLimit* and *noFirstOrder* are used instead.

In the violet colored box “Losses” one lookup table of the type *CombiTable2D* from the MSL is used for each loss type. By knowing the actual speed and torque, the operating point is identified and the losses of this point can be determined with the tables. Afterwards the electrical power is calculated from the sum of the losses and the measured mechanical power. This allows the calculation of the electrical current that must flow by measuring the voltage at the DC connectors.

## 4.3 Comparison Results

The hardware, software and solver settings which were used for the measurement of the computation time and the required time for map generation are summarized in Table 3.

### 4.3.1 Computation Time for Map Generation

The required computation time for the generation of characteristic maps was tested for each machine type for the two different map resolutions  $15 \times 30$  and  $30 \times 60$ . In Table 4 the results are shown. The column “unsim.” notes the number of skipped simula-

<sup>2</sup>The element is intended for the torque set machines to model the electric time constant. For torque controlled machines it is disabled per default, but it can be used to approximate the transfer function of desired to obtained torque.

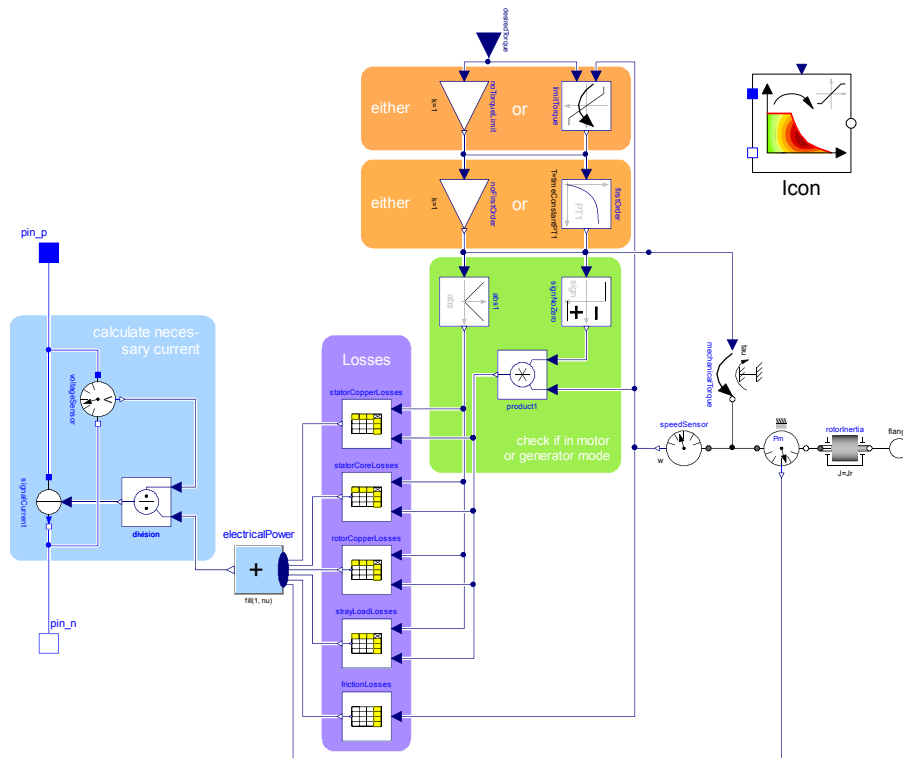


Figure 8: Model and icon for the torque controlled characteristic map

Hardware	Processor	Intel Core i7-2670QM
	RAM	8 GiB DDR3
Software	OS	Windows 7, 64 bit
	Dymola	2013 FD01, 32 bit
Settings	Solver	Dassl
	Tolerance	0.0001 (default)

Table 3: Used hardware, software and solver settings for time measurements

Type	tau	w	unsim.	req. time	time/sim.
ASM	15	30	204	190 s	0.78 s
ASM	60	30	854	740 s	0.78 s
PMSM	15	30	120	95 s	0.29 s
PMSM	60	30	488	370 s	0.28 s

Table 4: Required computation time (req. time), skipped simulations (unsim.) and required time for one simulation (time per sim.) for the generation of characteristic maps of different resolutions ( $\tau \times w$ )

tions, where the operating points lay above the maximum torque curve.

In the column “time/sim.” it is revealed that the required time for one simulation is constant for the different machine types. Hence, maps with 1000 calculated points are generated within 13 minutes for induction machines and less than 6 minutes for PMSMs. However, the map generation time is highly sensitive to the specified limits in which the losses and the torque have to settle. As it can be noticed in Figure 6 for the plot “torque settled” to low margins can result in chattering depending on the operating point. Hence, the high number of events slows down the map generation process significantly. So far no effort was spent to further investigate on that effect, but it would be well worth the effort if map generation time is of impor-

tance.

From Table 4 one can tell that the solving of the ASM’s models is about eight times as computationally intensive than the PMSM’s. This is caused by either the higher complexity of the ASM’s model due to effects like slip that is not present in the PMSM or due to the more complex controller structure. The exact cause has to be further investigated in future efforts.

### 4.3.2 Accuracy of the Losses

To compare the losses of the map model with those of the physical model, the setup depicted in Figure 9 is used. With `desiredSpeed` and `desiredTorque` the operating point is set for the map based and the

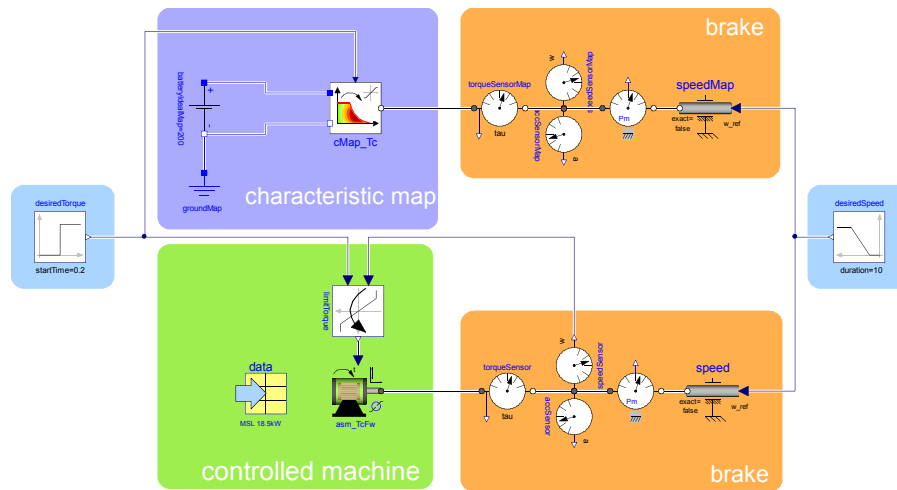


Figure 9: Used model for comparison of torque controlled map and physical machine

physical models and the input of the physical model is limited to the maximum torque curve. Similar models were used to compare the losses of the other machine types.

In Figure 10 the losses of an efficiency map are compared with the physical model. The desired torque is set on 100 N m, whereas the speed is increased from 0 rad s<sup>-1</sup> to 400 rad s<sup>-1</sup> within 10 s.

The comparison results of the torque controlled PMSM are shown in Figure 11. As one can see, the losses match very well. The plot “Deviation of total losses” at the bottom right shows the difference of the total losses calculated with the physical model and with the map model. Except of the negative peak at the beginning, the total deviation remains below 18 W. With the total losses starting at 1000 W and reaching about 1800 W when the highest deviation is observed the maximum error is 1%. The peak is caused by the stator core losses, since the current controllers set the maximum possible voltages at the beginning to obtain the desired torque. As the core losses are plotted over speed, this peak is hidden behind the axis to the left at 0 rad s<sup>-1</sup>.

### 4.3.3 Simulation Time

To compare the computation time of physical and map-based models, the model shown in Figure 12 is used. A sinusoidal or a trapezoid torque is requested from the machine and a speed dependent torque is used as load. With the integrator the total loss energy is obtained, which allows to check for the global error of the most interesting quantity at the end of the simulation. This error is computed between physical and ta-

ble based model, not the analytic solution as the word "error" may indicate.

The results for the sinusoidal input are depicted in Table 5. For every machine type the map model and the physical machine were tested for a simulation time of 1000 s and for 10000 s. Along with the required simulation times the speed improvement factor is given. In addition, the loss energy at the end of the simulation and the error of the map model are specified.

One can see that improvements of the simulation time of more than a factor of 200 can be obtained, while the error is kept below 1%. The highest improvements are achieved for the induction machine, while the torque set PMSM is only accelerated by a factor of 7.

## 5 Conclusion

The presented method allows convenient use of controlled physical electric machine models including advanced functionalities, to generate data for map-based models in order to accelerate simulations with still accurate results.

With the map models speed improvements with factors starting from 7, reaching up to 230. However the computation times of the physical models are highly sensitive to the performed simulation, the input signal and to the tuning of the controllers. Hence, it is required to test the models within specific applications to figure out which factors are obtained in practice.

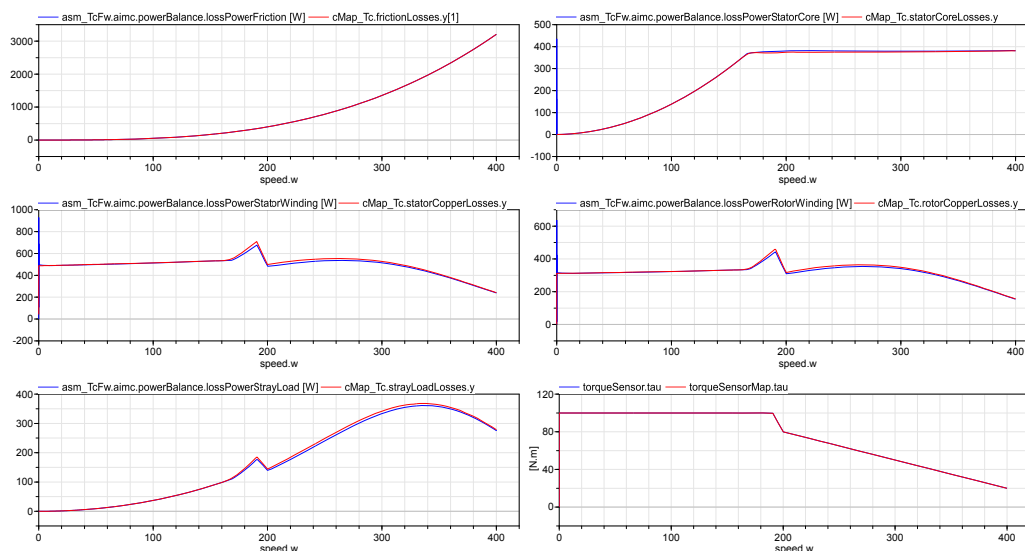


Figure 10: Losses of a physical induction machine model (blue) and characteristic map (red) with a resolution of  $15 \times 15$  for the motor mode

Type	CPU times			loss energy at simulation end		
	phys.	map	factor	phys.	map	error
ASM, 1000 s	82 s	0.35 s	234	1753.60 kJ	1763.04 kJ	0.5 %
ASM, 10000 s	807 s	3.5 s	230.5	17537.90 kJ	17633.20 kJ	0.5 %
PMSM, 1000 s	9.5 s	1.3 s	7.3	4778.35 kJ	4784.76 kJ	0.13 %
PMSM, 10000 s	95.7 s	13 s	7.4	47782.80 kJ	47848.70 kJ	0.13 %

Table 5: Required computation times for map based and physical models with sinusoidal input

## References

- [1] A. Haumer, C. Kral, H. Kapeller, T. Bäuml, and J. V. Gragger. The AdvancedMachines Library: Loss Models for Electric Machines. In *Proceedings of the 7th International Modelica Conference*, pages 847–854, September 2009.
- [2] J.-M. Kim and S.-K. Sul. Speed control of interior permanent magnet synchronous motor drive for the flux weakening operation. *Industry Applications, IEEE Transactions on*, 33(1):43–48, 1997.
- [3] C. Kral and A. Haumer. Modelica libraries for dc machines, three phase and polyphase machines. In *Proceedings of the 4th International Modelica Conference*, pages 549–558, March 2005.
- [4] M. Mengoni, L. Zarri, A. Tani, G. Serra, and D. Casadei. A Comparison of Four Robust Control Schemes for Field-Weakening Operation of Induction Motors. *Power Electronics, IEEE Transactions on*, 27(1):307–320, 2012.
- [5] P. Vas. *Sensorless Vector and Direct Torque Control (Monographs in Electrical and Electronic Engineering)*. Oxford University Press, USA, 1998.
- [6] J. Vinzenz, G. Harald, and G. C. Kral. The Smart-ElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives. In *Proceedings of the 5th International Modelica Conference*, pages 571–577, 2006.

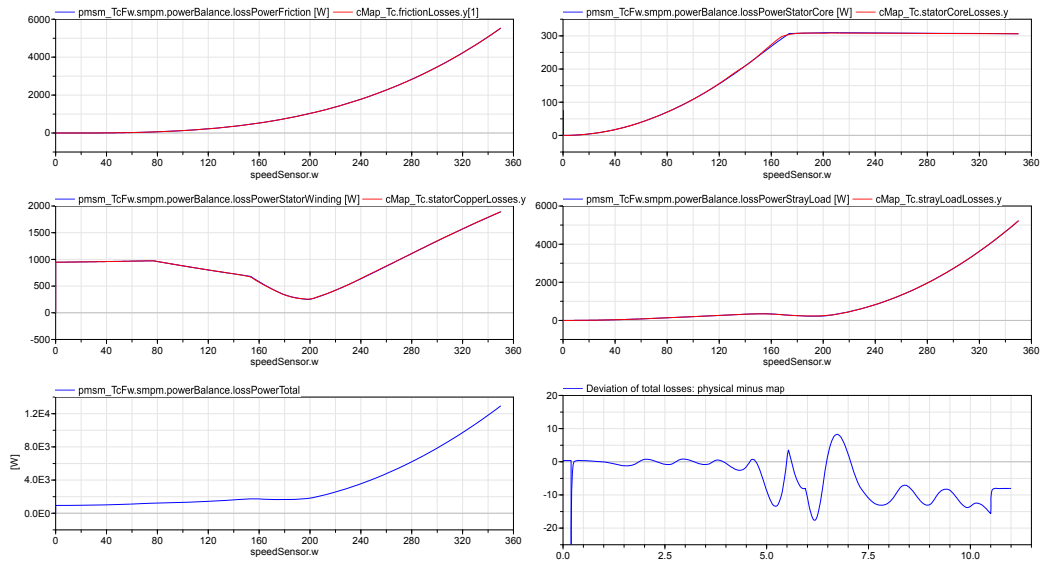


Figure 11: Losses of a torque controlled physical PMSM model (blue) and characteristic map (red) with a resolution of  $20 \times 15$  for the motor mode

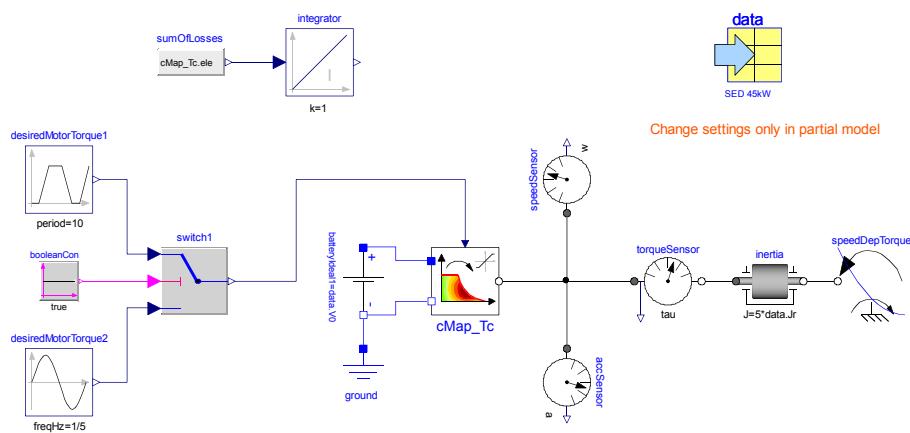


Figure 12: Model for comparison of computation times



# BuildSysPro: a Modelica library for modelling buildings and energy systems

Gilles Plessis

Aurélie Kaemmerlen

Amy Lindsay

EnerBaT – EDF R&D

Site des Renardières, 77818 Moret sur Loing CEDEX, FRANCE

gilles.plessis@edf.fr

aurelie.kaemmerlen@edf.fr

amy.lindsay@edf.fr

## Abstract

This paper presents the BuildSysPro Modelica library developed by the department of Energy in Buildings and Territories (EnerBaT) of EDF R&D. After a description of the library's structure and content, BESTEST validation results and a use case are presented.

This library is designed to be used in several contexts including building physics research, global performance evaluation, technology development and impact assessment. It is also a basis for urban and building stock simulation. BuildSysPro is intended for a relatively large audience ranging from R&D scientists to building services engineers.

BuildSysPro contains classes to describe the whole building and its energy systems including envelope components, HVAC systems and other energy conversion devices (DHW, thermal and photovoltaic panels...) and boundary conditions models. The models are designed for static and dynamic use, and for the representation of 0D/1D pure thermal and fluid dynamics. BuildSysPro in its current version contains around 380 models and 130 functions.

*Keywords: Modelica library; Building; Dynamic simulation; Numerical validations, Energy system*

## 1 Introduction

Since the building sector is one of the main energy consumers nowadays, energy policies drive existent and new buildings towards better performances. These evolutions raise quantity of questions regarding their ability to ensure the occupants' health and comfort while decreasing energy consumption and increasing energy efficiency. These questions rely strongly on multi-domain representations including thermal, electrical, hydraulic or chemical processes. Modelica being an object-oriented, equation based

language, is therefore well suited to represent this kind of coupled problems and complex systems.

The EnerBaT department of EDF R&D developed its own Modelica library, BuildSysPro, in order to perform multi-scale and multi-domain modelling. The choice of a new library was dictated by research needs, very specific for an energy producer and retailer, since they cover many domains.

BuildSysPro provides a comprehensive set of elementary 0D/1D components to describe envelope components, energy equipments and devices, and control systems. It is principally based on two branches of physics: pure thermal and thermo-fluid dynamics modelling. These classes are compliant with the *Thermal.HeatTransfer* and *Media* packages of the Modelica standard library to ensure a good level of interoperability with other Modelica libraries. These models are designed for static and dynamic modelling and can be used to create a whole building and its energy systems.

The BuildSysPro library has already been successfully used in several studies including:

- Technology performances and impact assessment [1][2],
- Sensitivity analysis regarding experimental validation [3],
- Urban simulation [4].

As ThermoSysPro, the EDF Modelica library for modelling power plants, BuildSysPro is developed under Dymola environment but is intended to be tool neutral. The BuildSysPro library is currently only available within partnership projects, but an open version is under construction.

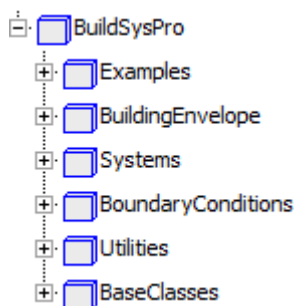
This paper is an overall presentation of BuildSysPro, focusing on the structure and some key elements of the library such as building envelope components, boundary conditions and HVAC systems. The key components of BuildSysPro being based on models from the Modelica standard Library, they are not described in detail. A focus is then made on validation through numerical comparisons with the IEA

BESTEST procedure. Finally the use of BuildSysPro is described on a basic use case aiming at analysing the matching between heat demand and supply in a residential building. For more complex applications and validations, the readers are welcome to read the papers [2], [3] and [4].

## 2 BuildSysPro Overview

### 2.1 Structure

The way of modelling building energy systems with BuildSysPro is similar to the approach commonly used by the building science community. On one side, the building envelope is mainly considered as an energy consumer and on the other side the energy systems and equipments are considered as producers. Figure 1 shows the top-level structure of BuildSysPro. This structure is quite similar to the Modelica standard library. It contains the usual *Examples*, *Interfaces*, *Components*, and *Utilities* packages at different hierarchical levels.



**Figure 1: Structure of BuildSysPro**

At the top level, the *Examples* package contains some reference buildings, including the Mozart house, which is a medium size detached house from a typological study of the French housing stock.

The *BuildingEnvelope* class is intended to describe the building envelope and provides components in a pure thermal or thermo-fluid approach. It also contains generic models of zones which can represent an entire building or a single room.

The *Systems* class is composed of five sub-packages. The *Controls* package provides control and regulation components for HVAC systems or energy equipments. The *Production*, *Distribution* and *Emission* packages provide components to design energy systems including HVAC systems or other equipments such as PV systems. This package also contains a *Utilities* package which provides for instance pre-processors to estimate system parameters from manufacturer data.

The *BoundaryConditions* package contains several models which offer the possibility of reading and pre-processing boundary conditions from files, such as weather data or normative indoor scenarios.

The *Utilities* package includes special Modelica *types*, *records*, *package icons*, *functions*, *blocks* and *models*. The *records* are used to set the parameters of various models in a hierarchical way (wall layers, walls, zones...). A *Math* sub-package contains, inter alia, some non linear solvers. A *Comfort* package includes some basic classes to describe human comfort in a room.

The *BaseClasses* package establishes the link with the Modelica standard library. It contains the same connectors as the *Modelica.HeatTransfer* and *Modelica.Media* packages. It also includes some other elementary models which are not of interest for end-users.

### 2.2 Interfaces

The interfaces of BuildSysPro are based on those from the Modelica standard library to ensure the compatibility of modelling. For instance, the connectors of the *HeatTransfer* class are based on two variables, a temperature as a potential and a heat flow rate as a flow. The *Fluid* class is compliant with the *Modelica.Media* class, that is to say a media model is described with the *Modelica.Media.Interfaces* and a connector similar to the *Media.Examples.Tests.Components.FluidPort*, which does not use stream connectors.

These interfaces were chosen to ensure modular and scalable approaches to model an entire building with its systems and equipments as well as single components or small districts, in pure thermal or in a thermo-fluid approach. Indeed pure thermal modelling can be accurate enough to predict the annual energy consumption due to heating, but not to design an air ventilation system nor to model pollutant transport.

### 2.3 Thermal zone description

Figure 2 illustrates the structure of the *BuildingEnvelope* package.

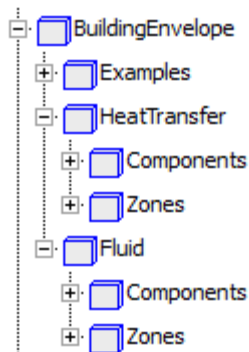


Figure 2: BuildingEnvelope package

One of the key elements of this package is the thermal wall model. It represents a 1D discrete multi-layer wall with several connectors for boundary conditions. A diagram view of this model can be seen on Figure 3. By convention, the right hand side corresponds to the inside whereas the left hand side can be both, inside or outside boundary conditions.

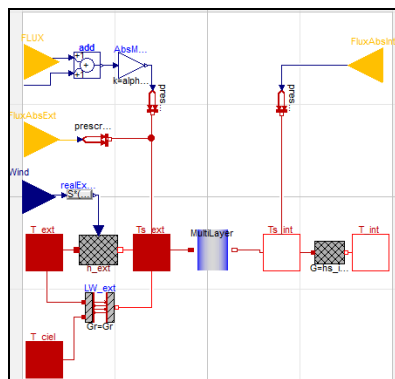


Figure 3: Diagram of the wall model

The blue component in the middle describes the conductive part of the wall. It is based on thermal conductors and capacitors connected in order to represent layers of homogenous material.

The causal connectors represented by yellow triangles are used to convey short wave radiations such as solar irradiance or transmitted solar radiation coming from the windows or the environment. They include either the cosine of the incidence angle, diffuse and direct flux or global flux.

The heat ports connect the model to the surrounding temperatures. Thanks to optional models and connections, convective heat transfers are considered with an  $h$  coefficient either fixed or controlled by wind speed. In the same way, the long wave radiative heat transfer is represented either with a fixed coefficient or with the Stefan–Boltzmann law using dry bulb and sky temperatures. The parameters of the wall model can be easily set thanks to various records. For instance, the parameters of the conductive part use a replaceable *WallType* record which contains the information described in Figure 4.

```

record WallType
  "Generic record for the conductive part of a wall"
  parameter Integer n=3 "Number of layers";
  parameter Integer[n] m=fill(1,n)
  "Number of nodes per layer (outside toward inside)";
  parameter Modelica.SIUnits.Length[n] e=0.2*fill(1,n)
  "Layer thicknesses (outside toward inside)";
  parameter BuildSysPro.Utilities.Records.MaterialProperties mat[n]
  "Material properties (outside toward inside)" a;
  parameter Integer[n] insulationPosition=zeros(n)
  "Insulation ? (outside toward inside)";
  a;
end WallType;
    
```

Figure 4: WallType record

A typical one-zone thermal model would be essentially composed of walls, one air node and air renewal. Figure 5 presents this simple thermal zone using combined convective and radiative heat transfers (without taking into account the wind speed or the sky temperature). Thus, depending on the assumptions considered, other types of thermal zones can be designed. For instance, instead of distributing the transmitted solar radiation onto the floor, other weighting methods can be used depending on the solar absorption coefficients and surface areas or view factors, as in the BESTEST calculations.

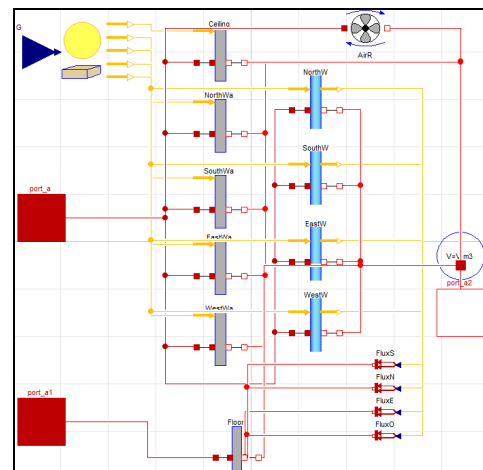


Figure 5: Diagram of a simple thermal zone

## 2.4 Systems and equipments

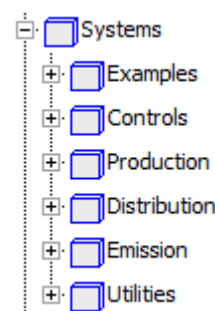


Figure 6: Systems package

As previously said, the *Systems* class is composed of five sub-packages mainly providing control strategies, production, distribution and emission compo-

nents. Some of these components are declined in different modelling levels adapted to different case studies: annual, daily, hourly or even sub-hourly time steps for prototyping and precise control.

For example, a simple convector can be used for studies on the electrical grid considering the 220V-50Hz power supply and the electronic regulation and components of the convector. Conversely, it can be modelled as an ideal heater or with chrono-proportional controls closer to the real dynamic behaviour for hourly outputs and human comfort studies.

Another example is thermodynamic systems, such as heat pumps, modelled in BuildSysPro either with an idealised coefficient of performance, or with empirical formulations depending on the boundary conditions (as in the use case described in paragraph 4), or with detailed modelling of the vapour compression and refrigeration cycle.

## 2.5 Boundary conditions

The boundary conditions describe the conditions on both sides of the building envelope. Figure 7 illustrates the structure of the *BoundaryConditions* package.

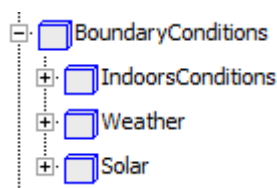


Figure 7: *BoundaryConditions* package

On the outdoor side, the weather data are applied with special treatment for solar data, and on the indoor side a temperature set point or other occupancy schedules are applied.

The weather data reader model requires a file that mainly contains the outdoor dry air temperature, the sky temperature, the relative humidity of the air, the wind data (speed and direction) and two solar radiations amongst diffuse horizontal, global horizontal, direct horizontal and direct normal. Inside the weather data reader model, the different missing fluxes are computed along with the position of the sun.

These weather solar data are then treated by models in the *Solar* package to obtain the incident direct and diffuse solar flux on the different surfaces, allowing a gain in computation time, especially for multi-zone modelling. Furthermore and as previously shown, specific yellow interfaces are included in BuildSysPro in order to graphically differentiate the solar boundary conditions in the model diagrams.

Figure 8 shows an assembly of the weather data reader and a boundary conditions model for shaded windows. The model in the middle computes incident solar radiation on a window under a solar mask from weather data.

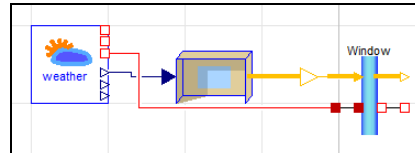


Figure 8: Example of the use of boundary conditions models – Shaded window

## 3 Validation cases

### 3.1 Introduction to validation methods

Software products can be validated with three complementary methods: analytic solution, empirical validation and comparative tests [5]. The first method is well suited for elementary models such as conductive transfer in walls for which an analytical solution is known. Empirical methods and experimental data are essential for the validation of complex models, but are difficult to analyse since there is a lot of uncertainties. Comparative tests are more repeatable and standardized tests are still carried out by the scientific community for the validation of different types of software. Thus, we have the ASHRAE standards in the USA and ISO norms in Europe.

Until 2010, the department of Energy in Buildings and Territories of EDF R&D mostly used CLIM2000 software [6] for research purposes. Before integrating new models, BuildSysPro was a quickly rebuilt version of CLIM2000 in Modelica language, possible since both approaches are very close:

- Variable time step solver,
- Equation-based modelling, describing the physical laws,
- Acausal connectors (heat ports),
- Graphical interface where elementary models are disposed and connected together.

CLIM2000 was validated with experimental measurements and comparative tests such as the International Energy Agency Building Energy Simulation Test and Diagnostic Method (IEA BESTEST). The results obtained with these validations were accurate and BuildSysPro was first validated by comparing the results of elementary models (conduction through walls, transmission through windows ...) with those of CLIM2000.

As a first approach, we will focus on the comparative BESTEST validation of BuildSysPro. This procedure is adapted to validate simple building components and has been previously applied to other libraries (e.g. [10]). As such, it is a good starting point in the validation of a new library.

### 3.2 Comparative tests: the BESTEST qualification procedure

The BESTEST method consists in a benchmark procedure for dynamic building energy simulation software established by a set of reference programs [7] [8]. This procedure was applied to validate the building envelope model of BuildSysPro, as has been done with CLIM2000 or other software and more recently some Modelica libraries [9] [10]. The entire procedure is described in [8], so we will only give the key points here.

The BESTEST procedure starts with a basic building envelope controlled in temperature with only one thermal zone. The envelope is derived into two inertia classes: light and heavy weight. This building is located in Denver, with an extreme climate: cold clear winters and hot dry summers. The building envelope is very sensitive to the external boundary conditions.

Small variations are made to this envelope, its control strategies and its internal gains, so that it might help in identifying deficient parts of the models thanks to diagnostic flow diagrams. A program successfully passes the validation if the required outputs are positively compared with the reference programs' outputs. Table 1 summarizes the base cases (600 to 650 and 900 to 950) and the free-floating cases (600FF to 950FF) since other cases defined in BESTEST are only used for diagnostics.

CASES	Opaque surface		Windows			Intern. gain [W]	Scenario H, C, V	Infiltration ACH	
	Type	$\alpha$ int/ext	$\epsilon$ int/ext	[m <sup>2</sup> ]	Or. Shade				
600	BML	0.6	0.9	12	S	-	200	20, 27, -	
610					1mH				
620					6 / 6	E / W			-
630					1mHV				
640					12	S			-
650							Setback		
900	BMH	0.6	0.9	12	S	-	200	20, 27, -	
910					1mH				
920					6 / 6	E / W			-
930					1mHV				
940					12	S			-
950							Setback		
600FF	BML	0.6	0.9	12	S	-	200	-,-,-	
650FF									-,-,-
900FF	BMH	0.6	0.9	12	S	-	200	-,-,-	
950FF									-,-,-

Table 1: Properties changed through the BESTEST cases

### 3.3 Modelling hypotheses

First of all, Figure 9 shows the graphical view of the BESTEST zone with a shaded south-oriented window. This diagram is more complex than the basic room shown in Figure 5 since more physical phenomena were considered. The reading of such an assembly is not yet user-friendly since the models were constructed with many conditional interfaces depending on the physical complexity chosen for the modelling. In the future, the data connectors (wind, temperatures, solar information, ...) will be grouped together with a Modelica Bus and this information will be used or not by the models.

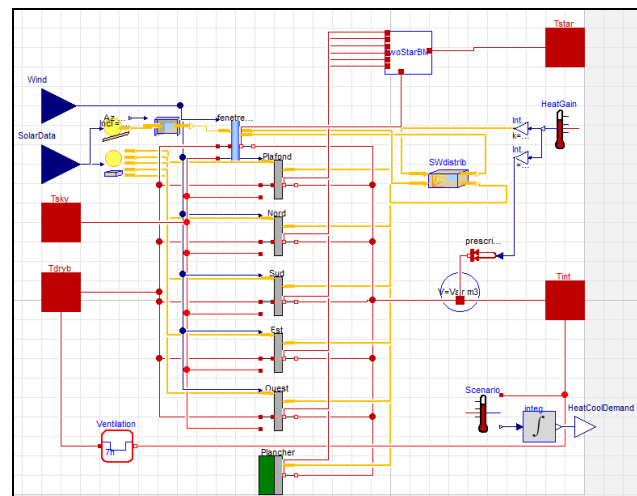


Figure 9: Graphical view of the South-shaded base model for the BESTEST in Dymola

We will not give an exhaustive list of the modelling hypotheses since there are many BESTEST specifications. We will only give the key points as to how the weather data were obtained and what were the hypotheses taken for modelling the envelope and the radiative heat transfers. Some models were improved from CLIM2000 to go through the BESTEST cases, since in the CLIM2000 version used for the BESTEST validation, a combined coefficient for convection and radiation was used and there were no shading devices.

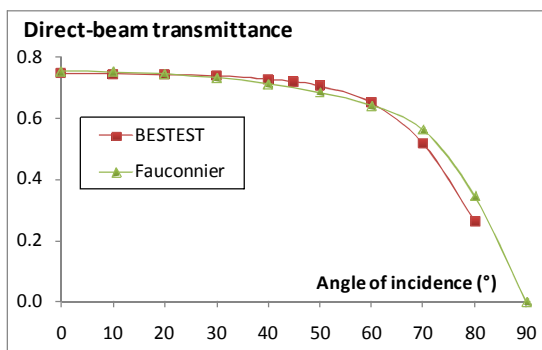
#### Weather data

The weather data file TMY for Denver was read before the simulations to extract the necessary inputs: direct normal and diffuse horizontal radiations, dry bulb and sky temperatures and wind speed. The data were read in local time with a delay of 30 minutes according to BESTEST reports. The sky temperature was obtained by a pre-processing using the Black-body model of the "Buildings" Modelica library [11]

which estimates it from total opaque sky cover, dry bulb temperature and dew point temperature.

### Envelope modelling

For conductive heat transfers, the walls are modelled with discrete element schemes as described in paragraph 2.3. The windows are modelled with a simple thermal conductor obtained from the U-value minus the convective and radiative parts of the heat transfer. The optical properties of the windows were estimated from the g-value and heat gain coefficient at normal incidence [12] given in the BESTEST specifications. The diffuse and direct properties are then very close to the ones recommended as seen in Figure 10 for direct beam transmittance.



**Figure 10: Angular dependence of the direct-beam transmittance of the double-pane window**

Convective heat transfer is modelled with a constant coefficient for the interior surfaces and with a correlation function of wind speed and surface texture for the outside.

### Radiation modelling

The solar irradiation is estimated with a clear-sky model with a separation between direct and diffuse radiations. The quantities transmitted into the room are redistributed as described in the BESTEST specifications using the view factors, surface areas and infrared emissivities.

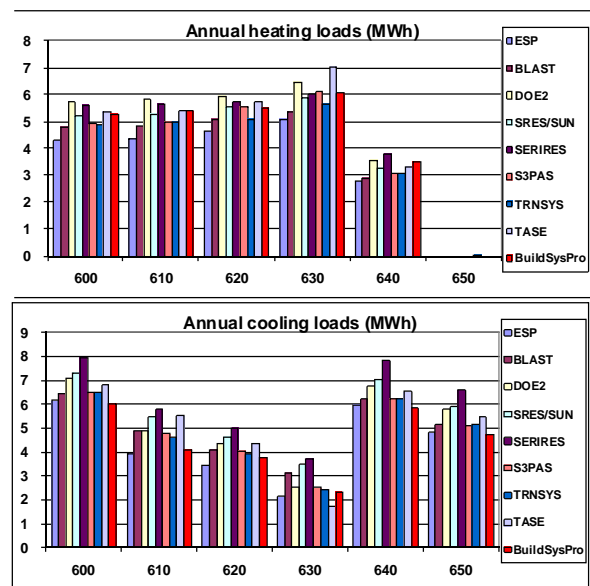
Longwave radiative heat transfers between wall surfaces and the outside environment are modelled using the corresponding temperatures (dry bulb and sky temperatures) and the sky and ground view factors for tilted surfaces. On the inside, a Two-Star model is used so that each surface exchanges with a virtual blackbody characterised by its radiative temperature [13].

## 3.4 Results of the qualification test

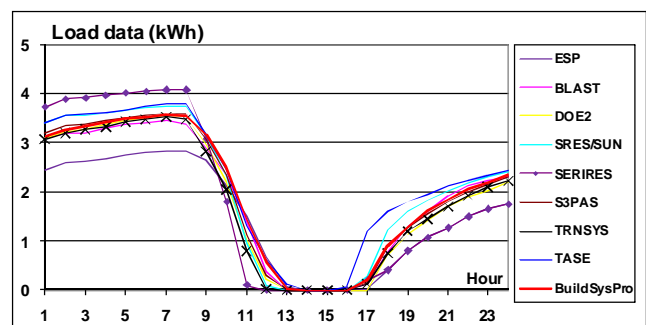
All the cases indicated in Table 1 were simulated with BuildSysPro. The figures below show only a

few representative results since the BESTEST method gives many outputs. The required outputs were integrated over the past hour as asked, especially since we used a variable time-step solver (DASSL). There are two kinds of outputs – annual and daily – as indicated in the list below:

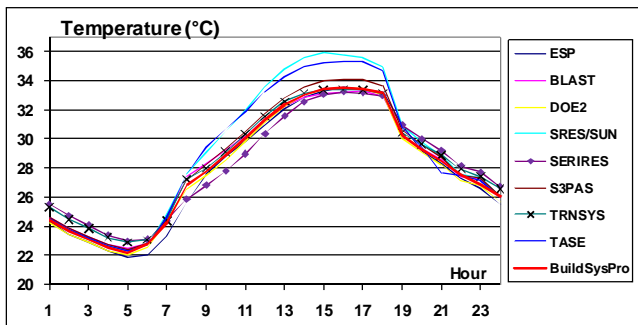
- Annual heating and cooling loads (See Figure 11 for low-mass cases 600 to 650) and annual hourly-integrated peak loads.
- Annual hourly-integrated maximum and minimum temperatures for free-floating cases and annual hourly 1°C temperature bin frequencies.
- Annual unshaded incident solar radiation, annual transmitted solar radiation (shaded and unshaded).
- Hourly heating and cooling loads (See Figure 12 for high-mass case 900).
- Hourly free-float temperature (See Figure 13 for high-mass case 950FF).
- Hourly unshaded incident solar radiation (See Figure 14 for South and West radiation on a clear day).



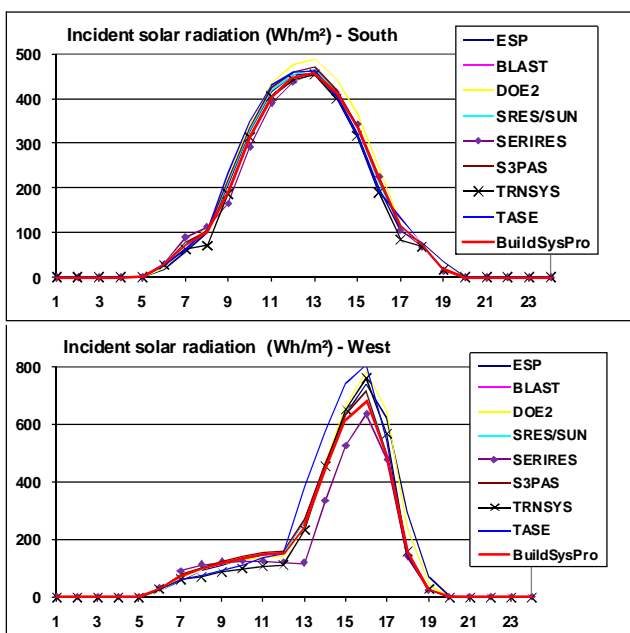
**Figure 11: Heating and cooling loads for lightweight cases**



**Figure 12: Hourly heating and cooling loads for high-mass case 900**



**Figure 13: Hourly free-float temperature for high-mass case 950FF**



**Figure 14: Hourly incident solar radiation – South and West on a clear day (July 27)**

In conclusion to all these test-cases, BuildSysPro obtains comparable results with the reference programs of the BESTEST qualification procedure. This validates the modelling approach of the library and the implementation in Modelica.

## 4 Use case

The use case presented in this section illustrates building energy analysis which can be easily done thanks to BuildSysPro. It is based on previous work [14].

### 4.1 Introduction

In low energy buildings and passive houses, the free available energy from the building's surroundings

should cover a significant share of the energy demand unlike common buildings for which it is mainly covered by HVAC systems. As a consequence, the heating and cooling demands are reduced and, most of the time, fluctuate around 0 W. Such a system should not be assessed based solely on energy consumption indicators; a dynamic analysis should also be conducted. This kind of study may allow the investigation of performance degradation due to part-load operation or energy storage potential.

### 4.2 Building envelope

For this use case, the “Mozart” house is selected. It is a detached home, one of the most representative of the French housing stock. The “Mozart” house is considered as medium size with 100 m<sup>2</sup> of living area and an air volume of 252.15 m<sup>3</sup>. The building is described in a low energy configuration. The U-values of the different envelope components are relatively low compared to the French housing stock.

The house is represented as only one thermal zone however internal walls are modelled and therefore contribute to the thermal inertia. The short wave radiations transmitted inside the building envelope through the windows are entirely absorbed by the floor. The long wave radiative heat transfers are taken into account through a combined heat transfer coefficient.

### 4.3 Boundary conditions

The building envelope is studied in a temperate climate, more precisely with the weather data from Trappes located near Paris in France. The weather reader model provides the outdoor dry air temperature, the direct and diffuse solar radiations and the sky temperature. The humid air and the wind data (speed and direction) are not used in this simple case. A fixed set point of 19°C for the indoor air temperature is used. For the sake of simplicity, no internal heat gains were taken into account.

Considering these boundary conditions, the annual heat demand for the building envelope is around 20 kWh.m<sup>-2</sup>.year<sup>-1</sup>.

### 4.4 HVAC systems

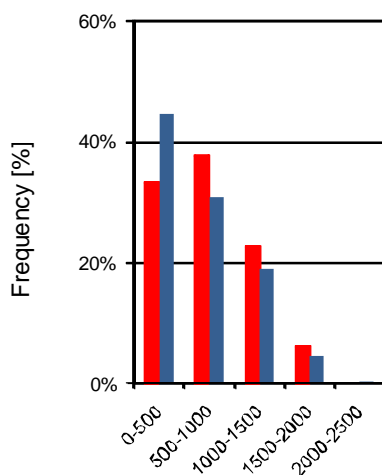
The HVAC systems are composed of a mechanical ventilation system and an intermittently controlled heat pump. This is an air to water heat pump which supplies hot water to a radiant heating floor. Under nominal conditions, the heat pump provides 2400 W of heat at a COP of 4.2. A dynamic empirical model

is used to represent the heat pump. It takes into account the operating conditions, the part load degradation and minimal operating time. The ventilation system is represented by a static model with an air change rate of  $0.35 \text{ vol.h}^{-1}$ .

## 4.5 Results

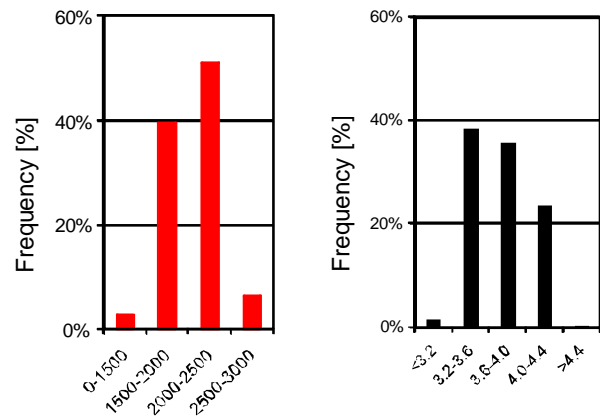
As previously mentioned this use case shows the advantage of using Modelica/Dymola and BuildSysPro for a simple building energy analysis.

First, the energy demands can be obtained throughout the year. Figure 15 shows the relative frequencies of the hourly heating and cooling demands, red and blue bars respectively. This figure provides statistical data about the structure of the energy demand for this specific building and these boundary conditions. This information is widely used for sizing applications or investigating the potential of a thermal storage.



**Figure 15: Relative frequencies of the hourly heating and cooling demands [Wh]**

However, one important matter with building energy systems is the ability for the HVAC systems to match the energy demand. As previously mentioned, for low energy buildings, the heating and cooling demands may fluctuate around  $0 \text{ W}$ , far away from nominal conditions, and therefore decrease the system's performances. Figure 16 (a) and (b) illustrate the relative frequencies for the hourly heat supply and the COP and give information on the heat pump's effective operating conditions.



**Figure 16: Relative frequencies of (a) heat supply [Wh] (b) COP [-]**

The discrepancies between Figure 15 for the hourly heating demand and Figure 16 (a) indicate that the intermittently controlled heat pump is not adapted for this building; a variable-speed controlled heat pump would be more suited. It induces a decrease in performances which can be observed on Figure 16 (b) compared to the nominal COP of 4.2.

All the figures presented in this section were obtained thanks to a post-processing of the simulation result files. BuildSysPro provides some basic functions for result export and analysis; however it is not intended to provide a comprehensive set of post-processing methods.

## 5 Conclusions

The BuildSysPro library has demonstrated its capabilities to model complex buildings and energy systems. It is a solid solution to two initial problems of our work:

- designing a library of models able to deal with advanced scientific problems related to low-energy buildings
- and providing a global solution for the modelling and simulation needs of the EnerBaT department

The knowledge and experience derived from our previous research tools have been integrated in BuildSysPro. It is developed with a goal of simplicity, yet is comprehensive enough to be used for building physics and technology development as well as global performance assessment or prospective simulation, making it the reference modelling and simulation tool for the entire EnerBaT department as of today.

BuildSysPro is also built with a multi-domain and multi-scale approach, as it is able to handle purely thermal problems as well as coupled multiphysics,



ranging from the basic envelope component to the entire building or building stock. In addition, some use cases have confirmed the strength of the modular approach for quick studies.

Moreover, the envelope modelling of this library has been validated thanks to the IEA BESTEST qualification method. Though this approach is basic and quite far from the complexity of a real building, it is a necessary step in the evaluation of building models. Indeed, a Modelica library for modelling buildings that does not pass the BESTEST validation cases would be considered as untrustworthy.

Upcoming work will include specific validation campaigns that will be conducted with experimental data from European laboratories.

Research in progress and future developments will also use other capabilities of Modelica such as FMI for co-simulation or hardware-in-the-loop applications in addition to the current work around low-energy building envelope and systems.

## References

- [1] Blervaque, H., Filfli, S., Stabat, P., Schumann, M., Marchio, D., Comparative Analysis of Air-To-Air Heat Pump Models for Building Energy Simulation. Proceedings of SimBuild 2012.
- [2] Lindsay, A., Energetic Evaluation of an Active Cooling System for Building Integrated Photovoltaics. Proceedings of EU PVSEC 2013.
- [3] Bontemps, S., Kaemmerlen, A., Blatman G., Mora L., Reliability Of Dynamic Simulation Models For Building Energy In The Context Of Low-energy Buildings. Proceedings of IBPSA 2013.
- [4] Kim, E., Plessis, G., Roux, JJ., Hubert JL., Reduction Of Building Models For Use In Urban Energy Analysis. Proceedings of IBPSA 2013.
- [5] Roujol S., building energy simulation software package uncertainty and validation, PhD Thesis, Mines ParisTech, 2003.
- [6] Murphy, K.M. and Deque, F., An open ended modular interface and controller library for CLIM2000, International Building Performance Simulation Association, 1997.
- [7] Judkoff R. and Neymark J., Model validation and testing: the methodological foundation of ASHRAE Standard 140, ASHRAE Transactions 112(2). Atlanta: GA: American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2006. 367-37.
- [8] Judkoff, R., Neymark, J., International Energy Agency Building Energy Simulation Test (IEA BESTEST) and Diagnostic Method, NREL/TP-472-6231. Golden, CO: NREL, 1995
- [9] Yuan S. and O'Neill Z., Testing and validating an equation-based dynamic building program with ASHRAE standard method of test. In: Proceedings of the 3rd SimBuild Conference, Berkeley, USA, 2008.
- [10] Videla, J. I. and Lie, B., A New Energy Building Simulation Library. Proceedings of Modelica 2006
- [11] Wetter M., Zuo W., Noudou T. S., Recent Developments of the Modelica Buildings Library for Building Energy and Control Systems, Proceedings of the 8th International Modelica Conference, 2011.
- [12] Covallet D., Woloszyn M. et Greffier V., Apport des vitrages actifs pour le confort d'été, Proceedings IBPSA France, 2006
- [13] Felgner F., Agustina S., Cladera Bohiga R., Merz R., Litz L., Simulation of Thermal Building behaviour in Modelica, Proceedings of the 2<sup>nd</sup> International Modelica Conference, 2002
- [14] Filfli, S., Bouia, H., Simulation avec Modelica du fonctionnement d'une PAC dans un bâtiment BBC : impact d'un pas de temps infra-horaire. Proceedings of the CIFQ 2013.



# Efficient Numerical Integration of Dynamical Systems based on Structural-Algebraic Regularization avoiding State Selection

Lena Scholz    Andreas Steinbrecher  
Technical University Berlin, Department of Mathematics  
Str. des 17. Juni 136, 10623 Berlin

## Abstract

Differential-algebraic equations naturally arise in the modeling of dynamical processes, in particular using MODELICA as modeling language. In general, the model equations can be of higher index, i.e., they can contain *hidden constraints* which lead to instabilities and order reductions in the numerical integration. Therefore, a regularization or remodeling of the model equations is required. One way to obtain the required information on the hidden constraints is a structural analysis based on the sparsity pattern of the system. For the determination of a regular index-reduced system formulation then, usually, a crucial step is the so-called *state selection*. In this paper, we will present a new approach for the remodeling of dynamical systems that uses the information obtained from the structural analysis to construct a regularized overdetermined system formulation. This overdetermined system can then be solved using specially adapted numerical integrators, in such a way that the state selection can be performed within the numerical integrator during runtime of the simulation.

*Keywords: DAEs; regularization; structural analysis; overdetermined system; state selection*

## 1 Introduction

The MODELICA language is a common tool for modeling of dynamical processes. In general, the model equations that describe the dynamical process consist of differential equations in combination with algebraic constraints, i.e., we have to deal with so-called *differential-algebraic equations* (DAEs).

The solutions of such systems have to satisfy the algebraic constraints, but, in general, not all constraints are stated in an explicit way. In particular, if the resulting system of DAEs is of higher index there exist so-called *hidden constraints* and the numerical treatment leads to instabilities, inconsistencies and

possibly non-convergence of the numerical methods, see [2, 4, 6, 8]. Thus, a *regularization* or *remodeling* of the model equations is required to guarantee stable and robust numerical computations, see also [3, 6, 8, 15].

The current state of the art in many modeling and simulation tools to deal with high index DAEs is to use some kind of structural analysis based on the sparsity pattern of the system. Here, generic structural information is used to identify the constraints, to determine the index of the system, and to compute an index-reduced system model. Hereby, a crucial step is the so-called *state selection* that is required in order to introduce new algebraic variables (the so-called *dummy derivatives*) for the selected differential components of the DAE system in order to obtain a regular index-reduced formulation.

In this paper, we present a new regularization approach for the remodeling of dynamical systems that uses the information provided by the structural analysis, in particular by the Signature Method [12], to construct an overdetermined system regularization that can be solved using a specially adapted numerical integrator. This approach has the great advantage that the problem of state selection can be moved within the numerical integrator and can therefore be performed during the runtime of the simulation.

In the following, we consider *quasi-linear DAEs* of the form

$$E(x,t)\dot{x} = k(x,t), \quad (1)$$

on the domain  $\mathbb{I} = [t_0, t_f]$  with initial values  $x(t_0) = x_0 \in \mathbb{R}^n$ , where  $E \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{n,n})$  is called the *leading matrix* of the quasi-linear DAE and  $k \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^n)$  its *right-hand side*. Furthermore,  $x : \mathbb{I} \rightarrow \mathbb{R}^n$  represent the *unknown variables*. The DAE system (1) is assumed to be uniquely solvable and nonredundant. Furthermore, we assume that the rank of the leading matrix  $E$  is constant for

all  $(x, t) \in \mathbb{R}^n \times \mathbb{I}$  and that the rank of the partial derivatives of the (hidden) constraints with respect to  $x$  is constant for all consistent  $(x, t) \in \mathbb{R}^n \times \mathbb{I}$ . Note that in general these assumptions are not necessary and can be relaxed, see [16].

## 2 Structural Analysis of DAEs

In many simulation environments like DYMOLA, OPENMODELICA or MAPLESIM a structural analysis is used to reduce the index of the DAE system relying on its sparsity structure, e.g., there are various versions and extensions of *Pantelides Algorithm* [10] in combination with the *Dummy Derivative Approach* [9], or the *Signature Method* [12]. These structural approaches have the great advantage that fast and efficient linear optimization algorithms based on graph theoretical concepts can be used and further structural information like a block lower triangular form of the system can be extracted which is essential for efficient and fast computations.

In this section, we will shortly review the basic steps of the Signature Method ( $\Sigma$ -method) introduced in [12]. For ease of representation we write the model equations (1) as

$$F(t, x, \dot{x}) = 0 \tag{2}$$

with  $F(t, x, \dot{x}) := E(x, t)\dot{x} - k(x, t)$ , where  $F \in \mathcal{C}(\mathbb{I} \times \mathbb{R}^n \times \mathbb{R}^n, \mathbb{R}^n)$ , and we denote by  $F_i$  the components of the vector  $F$  and by  $x_j$  the components of the vector  $x$ . Then, the  $\Sigma$ -method consists of the following steps:

1. Built the *signature matrix*  $\Sigma = [\sigma_{ij}]_{i,j=1,\dots,n}$

$$\sigma_{ij} := \begin{cases} \text{highest order of derivative of } x_j \text{ in } F_i, \\ -\infty \text{ if } x_j \text{ does not occur in } F_i. \end{cases}$$

2. Find a *highest value transversal* (HVT) of  $\Sigma$ , i.e., a *transversal*  $T$  of  $\Sigma$

$$T = \{(1, j_1), (2, j_2), \dots, (n, j_n)\},$$

where  $(j_1, \dots, j_n)$  is a permutation of  $(1, \dots, n)$ , with maximal value  $Val(T) = \sum_{(i,j) \in T} \sigma_{ij}$ .

3. Compute the *offsets vectors*  $c$  and  $d$  with  $c_i \geq 0$  such that

$$\begin{aligned} d_j - c_i &\geq \sigma_{ij} \text{ for all } i, j = 1, \dots, n, \\ d_j - c_i &= \sigma_{ij} \text{ for all } (i, j) \in T. \end{aligned} \tag{3}$$

4. Form the  $\Sigma$ -*Jacobian*  $\mathfrak{J} = [\mathfrak{J}_{ij}]_{i,j=1,\dots,n}$ , with

$$\mathfrak{J}_{ij} := \begin{cases} \frac{\partial F_i}{\partial x_j^{(\sigma_{ij})}} & \text{if } d_j - c_i = \sigma_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

5. Built the *reduced derivative array*  $\mathcal{F}(t, \mathcal{X}) = 0$  consisting of

$$\begin{aligned} F_i(t, x, \dot{x}) &= 0, \\ \frac{d}{dt} F_i(t, x, \dot{x}) &= 0, \\ &\vdots \\ \frac{d^{(c_i)}}{dt^{(c_i)}} F_i(t, x, \dot{x}) &= 0 \end{aligned}$$

for all  $i = 1, \dots, n$  with

$$\mathcal{X} = [x_1, \dot{x}_1, \dots, x_1^{(d_1)}, \dots, x_n, \dot{x}_n, \dots, x_n^{(d_n)}]^T.$$

6. Success check: if the algebraic system  $\mathcal{F}(t^*, \mathcal{X}^*) = 0$  has a solution  $(t^*, \mathcal{X}^*) \in \mathbb{I} \times \mathbb{R}^{n+\sum_{i=1}^n d_i}$  and  $\mathfrak{J}$  is nonsingular at  $(t^*, \mathcal{X}^*)$ , then the  $\Sigma$ -method succeeds.

If the  $\Sigma$ -method succeeds, it allows to determine the *structural index* of the DAE as

$$v_S := \max_i c_i + \begin{cases} 0 & \text{if all } d_j > 0, \\ 1 & \text{if some } d_j = 0. \end{cases}$$

We call  $\mathfrak{J}$  the  $\Sigma$ -Jacobian since it is in general not the analytical Jacobian, but defined by the offset vectors. The HVT as well as the offset vectors can be computed efficiently by solving a *linear programming problem* (LPP) and the corresponding dual problem, see [12]. Note that usually there is not only one uniquely determined HVT, and also the offset vectors  $c$  and  $d$  are not uniquely defined by the conditions (3). However, there exists a unique element-wise smallest solution of the dual problem, the so-called *canonical offsets*, that is independent of the chosen HVT.

If the  $\Sigma$ -method succeeds for a given system (2) at a consistent point, the canonical offset vector  $c$  gives the required information which equations have to be differentiated and how many times in order to be able to extract all hidden constraints. Thus, the reduced derivative array  $\mathcal{F}$  can be obtained by adding the derivatives of  $F_i$  up to order  $c_i$  to the original system for all  $i = 1, \dots, n$ .

**Example 2.1** We illustrate the steps of the  $\Sigma$ -method for the example of the simple pendulum of mass  $m = 1$ ,

length  $\ell > 0$  under gravity  $\mathbf{g}$ , see also [12]. The system equations are given by

$$\begin{aligned} F_1(t, x, \dot{x}) &= \dot{p}_1 - q_1 &= 0, \\ F_2(t, x, \dot{x}) &= \dot{p}_2 - q_2 &= 0, \\ F_3(t, x, \dot{x}) &= \dot{q}_1 + 2p_1\lambda &= 0, \\ F_4(t, x, \dot{x}) &= \dot{q}_2 + 2p_2\lambda + \mathbf{g} &= 0, \\ F_5(t, x, \dot{x}) &= p_1^2 + p_2^2 - \ell^2 &= 0, \end{aligned} \quad (4)$$

with  $x = [p_1 \ p_2 \ q_1 \ q_2 \ \lambda]^T$ . The signature matrix for this system is given by

$$\Sigma = \begin{bmatrix} \boxed{1} & - & \boxed{0} & - & - \\ - & \boxed{1} & - & \boxed{0} & - \\ 0 & - & \boxed{1} & - & \boxed{0} \\ - & 0 & - & \boxed{1} & \boxed{0} \\ \boxed{0} & \boxed{0} & - & - & - \end{bmatrix},$$

where the two possible HVTs are marked by gray and blue boxes. (Here, the entry  $-$  stands for  $-\infty$ .) The canonical offset vectors are given by  $c = [1, 1, 0, 0, 2]$  and  $d = [2, 2, 1, 1, 0]$  (independently of the chosen HVT). The corresponding  $\Sigma$ -Jacobian is given by

$$\mathfrak{J} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 2p_1 \\ 0 & 0 & 0 & 1 & 2p_2 \\ 2p_1 & 2p_2 & 0 & 0 & 0 \end{bmatrix}$$

and the reduced derivative array takes the form

$$\mathcal{F}(t, \mathcal{X}) = \begin{bmatrix} \dot{p}_1 - q_1 \\ \dot{p}_1 - \dot{q}_1 \\ \dot{p}_2 - q_2 \\ \dot{p}_2 - \dot{q}_2 \\ \dot{q}_1 + 2p_1\lambda \\ \dot{q}_2 + 2p_2\lambda + \mathbf{g} \\ p_1^2 + p_2^2 - \ell^2 \\ 2p_1\dot{p}_1 + 2p_2\dot{p}_2 \\ 2p_1\ddot{p}_1 + 2\dot{p}_1^2 + 2p_2\ddot{p}_2 + 2\dot{p}_2^2 \end{bmatrix} = 0. \quad (5)$$

Thus, the  $\Sigma$ -Jacobian  $\mathfrak{J}$  is nonsingular at every consistent point and the  $\Sigma$ -method succeeds with  $v_S = \max_i c_i + 1 = 3$ .

The information provided by the HVT and the offset vectors can also be used to introduce new algebraic variables for selected differential variables yielding an extended square regularized system, for details see also [13]. However, it may happen that the success check of the  $\Sigma$ -method fails as can be seen in the following example.

**Example 2.2** Consider the simple DAE system

$$\begin{aligned} \dot{x}_1 &= x_3 + b_1 \\ \dot{x}_2 &= x_4 + b_2 \\ 0 &= x_2 + x_3 + x_4 + b_3 \\ 0 &= -x_1 + x_3 + x_4 + b_4 \end{aligned} \quad (6)$$

which is regular and of differentiation index ( $d$ -index) 3. If we apply the  $\Sigma$ -method to system (6), we get the signature matrix

$$\Sigma = \begin{bmatrix} \boxed{1} & - & 0 & - \\ - & \boxed{1} & - & 0 \\ - & 0 & \boxed{0} & 0 \\ 0 & - & 0 & \boxed{0} \end{bmatrix} \quad (7)$$

with marked HVT on the diagonal and canonical offset vectors  $c = [0, 0, 0, 0]$  and  $d = [1, 1, 0, 0]$ . The corresponding  $\Sigma$ -Jacobian is given by

$$\mathfrak{J} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & -1 & -1 \end{bmatrix} \quad (8)$$

and  $\mathfrak{J}$  is singular, i.e., the  $\Sigma$ -method fails.

Example 2.2 shows that for regular and therefore uniquely solvable systems the structural analysis can fail. In the following, systems for which the success check fails since the  $\Sigma$ -Jacobian is singular will be called *structurally singular*<sup>1</sup>. It has been shown in [13] that this is the case for certain coupled systems that are obtained by coupling semi-explicit  $d$ -index 1 subsystems, when the coupling results in redundancies or in an increase in the index. Nevertheless, the structural approach works well in many cases and for many important structures as e.g. systems in Hessenberg form, see [12].

**Remark 2.3** It has been shown in [12] that Pantelides Algorithm [10] and the Signature Method described above are essentially equivalent in the sense that if they can both be applied and they both succeed (or converge) they result in the same structural index and the offset vector  $c = [c_i]$  corresponds to the number of differentiations for each equation  $F_i$  as determined by Pantelides Algorithm. The advantage of using the Signature Method is the fast and efficient computation of the offset vectors via LPPs and the direct success

<sup>1</sup>Note that the term *structurally singular* is also used with a different meaning in other areas of research.

check (i.e., checking the regularity of the  $\Sigma$ -Jacobian) that allows us to use the results for further treatment. Note that Pantelides Algorithm will not converge in cases where the success check of the Signature Method fails.

### 3 Regularization using Overdetermined Formulations

Regularization approaches for high index DAEs like the Dummy Derivatives Approach [9] or index reduction by Minimal Extension [7] consist of adding the hidden constraints to the system equation and the selection of certain differential components that can then be replaced by new algebraic variables in order to lower the index of the system and to obtain a new regular index-reduced system formulation. Hereby, a problem is that the choice of states that are selected can change during the numerical integration (e.g., if the pendulum moves from the vertical to the horizontal position). Thus, if the state selection is performed outside the numerical integrator this often is computational inefficient. In the following, we will present a regularization of quasi-linear DAEs (1) that are of higher index, i.e., that contain hidden constraints. This regularization is based on an overdetermined system formulation in order to overcome the difficulties in the numerical simulation.

If the structural analysis presented in Section 2 succeeds, the offset vector  $c$  gives us the required information about the hidden constraints in the system. If the success check of the  $\Sigma$ -method fails, we can use the procedure proposed in [14, 15] to determine the hidden constraints of a quasi-linear DAE (1).

**Remark 3.1** For structurally singular systems in semi-explicit form arising in coupled systems of DAEs a combined structural-algebraic approach has been proposed in [13] that can be applied in cases where the success check fails, but nevertheless allows us to use certain information provided by the structural analysis. In this way, the determination of the hidden constraints can be improved.

Let us denote the hidden constraints by

$$0 = h(x, t), \tag{9}$$

where  $h : \mathbb{R}^n \times \mathbb{I} \rightarrow \mathbb{R}^c$  with  $\text{rank} \left( \frac{\partial h}{\partial x}(x, t) \right) = \text{const.}$  for all consistent  $(x, t) \in \mathbb{R}^n \times \mathbb{I}$ . Adding the hidden constraints to the quasi-linear DAE (1) leads to the

overdetermined DAE

$$E(x, t)\dot{x} = k(x, t), \tag{10a}$$

$$0 = h(x, t) \tag{10b}$$

consisting of the original quasi-linear DAE (1) and all hidden constraints (9). This overdetermined formulation (10) is equivalent to the original DAE (1) in the sense that both have the same solution set. Note that the unknowns  $x$  are unchanged, i.e., a transformation of the state variables is not necessary and the number of unknowns is not increased (in contrast to the dummy derivative approach). The overdetermined formulation (10) has the advantage that all constraints are stated in explicit form, i.e., no hidden constraints exist anymore. A further advantage of the overdetermined formulation (10) is the fact that it is not necessary to apply analytical manipulations for the determination of a square and uniquely solvable system of DAEs (provided that consistent initial values are given).

**Example 3.2** For the simple pendulum the hidden constraints can be derived from the reduced derivative array (5) and consists of

$$\begin{bmatrix} p_1^2 + p_2^2 - \ell^2 \\ 2p_1q_1 + 2p_2q_2 \\ -4p_1^2\lambda + 2q_1^2 - 4p_2^2\lambda - 2p_2g + 2q_2^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{11}$$

### 4 Numerical Approach

Unfortunately it is impossible to model and integrate the overdetermined formulation (10) within the common MODELICA frameworks. Therefore, the above described approach has been incorporated into a prototype MODELICA framework named MPSSim (Multi-Physics System Simulation). Here, a direct numerical integrator has been adapted for the overdetermined regularization (10).

In the following, the used adapted numerical integration scheme is exemplary illustrated for the implicit Euler method. In this case, the discretization of the overdetermined system (10) leads to the overdetermined nonlinear system

$$0 = \begin{bmatrix} E(x_k, t_k)(x_k - x_{k-1}) - \tau_k k(x_k, t_k) \\ -\tau_k h(x_k, t_k) \end{bmatrix} \tag{12}$$

to determine the next iterate  $x_k$ . Here  $\tau_k$  denotes the stepsize in the integration step  $k = 1, \dots, N$  in the Euler scheme,  $t_k$  the discrete time point, and  $x_k$  the approximation of the solution  $x(t_k)$  at the point  $t_k$ .

The nonlinear system (12) is no longer exactly solvable because of discretization and rounding errors during the numerical integration. Therefore, it is only possible to find an approximation  $\tilde{x}_k$  which minimizes the residual  $r \neq 0 \in \mathbb{R}^{n+n_c}$  with

$$r = \begin{bmatrix} r_D \\ r_C \end{bmatrix} = \begin{bmatrix} E(\tilde{x}_k, t_k)(\tilde{x}_k - x_{k-1}) - \tau_k k(\tilde{x}_k, t_k) \\ -\tau_k h(\tilde{x}_k, t_k) \end{bmatrix}$$

in a certain sense. In general, such an approximation results in a residual  $r_C \neq 0$ , which in turn leads to unfulfilled constraints, i.e.,  $0 \neq h(x_k, t_k)$ , not even within machine precision. This would lead to the typical difficulties in the numerical integration of higher index DAEs, i.e., instabilities, convergence problems, inconsistencies, or the solution drifts away from the original solution manifold.

In order to avoid these problems it is necessary to make sure that the constraints are always satisfied during numerical integration. This can be achieved if the nonlinear system (12) is treated separately such that the next iterate  $x_k$  satisfies the lower part, i.e., the constraints, exactly or within a prescribed precision, while  $x_k$  yields a minimal residual in the upper part, i.e., in the differential part. The described numerical approach is implemented in the software package QUALIDAES (QUAsi LInear DAE Solver). This software package is suited for the direct numerical integration of regularized overdetermined model equations and is based on the 3-stage implicit Runge-Kutta method of type Radau IIa of order 5, see [5, 6]. QUALIDAES is integrated as numerical solver into the MPSSim framework. In the current version the user has to provide the model equations already given in overdetermined regularized form (10) formulated as MODELICA model. Then, using the translator M02FOR [1] a FORTRAN source code is generated that can be used to solve the model equations with the solver QUALIDAES. The FORTRAN source code is automatically compiled and linked to the solver QUALIDAES. In Figure 1 the approach for the numerical treatment of models defined in MODELICA using MPSSim is illustrated. Note that within this framework, it is not necessary to determine a dynamic (state) selector, since this is achieved automatically within the separated treatment of (12) by its numerical solution, as described above. For a convenient usage also a graphical user interface (GUI) has been implemented in Matlab (see Figure 2) allowing the graphical representation of the obtained numerical results and can be used for further post-processing.

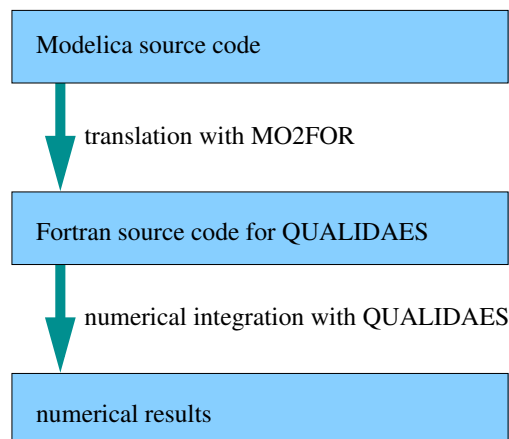


Figure 1: Scheme of MPSSim

## 5 Numerical Example

To show the promising performance of integrating a DAE system using MPSSim with an overdetermined system formulation we have compared the simulation of the simple pendulum equations given in Example 2.1 using MPSSim, MapleSim, Dymola and OpenModelica. In order to have a measurement for the error we include another equation in the system describing the total energy

$$E = \frac{1}{2}m(q_1^2 + q_2^2) + mgp_2$$

that should be preserved for all  $t \in \mathbb{I}$  and every solution of the system (4). We use a gravitational constant of  $g = 13.7503716373294544 \frac{m}{s^2}$  to ensure a time period of  $T = 2s$  for the motion of the pendulum and a mass of  $m = 1kg$  as well as a length of  $\ell = 1m$ . At first we simulate the system for  $t \in [0s, 100s]$  with given (fixed) consistent initial conditions

$$\begin{aligned} p_1(0) &= 1, & p_2(0) &= 0, & q_1(0) &= 0, \\ q_2(0) &= 0, & \lambda(0) &= 0, & E(0) &= 0, \end{aligned}$$

and a prescribed error tolerance of  $10^{-7}$  (for both the absolute and relative error). In the simulation with MPSSim we solve the overdetermined formulation (4) together with (11) containing all hidden constraints, while the other simulation tools use the original d-index 3 formulation (4) and the index reduction is performed within the tool using different index reduction strategies. The values  $E(t_f)$  of the total energy at the final time point  $t_f = 100s$  together with the required CPU times needed for the integration are listed in Table 1. In Dymola a modified version of the multi-step solver Dassl is used. Here, quite a large number of state selections are required (alternating selecting the

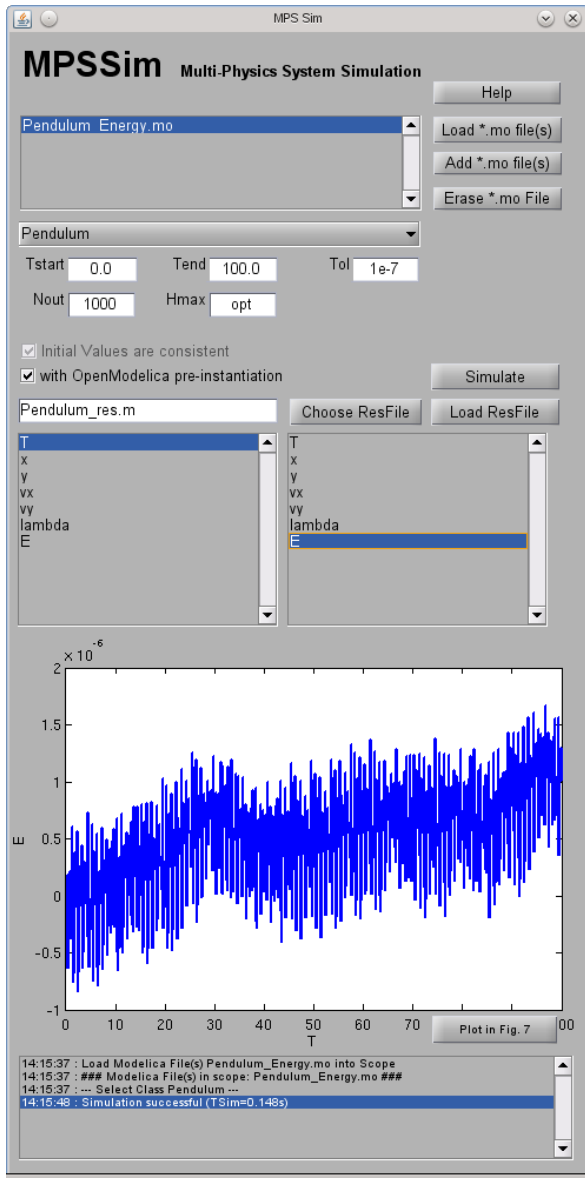


Figure 2: Matlab-GUI of MPSSim

states  $p_1$  and  $q_1$ , or  $p_2$  and  $q_2$ ). In MapleSim we use once the provided solver CK45 suited for semi-stiff problems, which is usually less accurate but faster, and once the Rosenbrock method suited for stiff systems, which yields a higher accuracy at the expense of more required CPU time. Furthermore, the corresponding results obtained for a long time simulation for  $t \in [0s, 1000s]$  are listed in Table 2. Note that OpenModelica fails to integrate the system in this case.

Comparing the obtained results one can see that the projection strategy onto the constraint manifold used within MapleSim yields an accurate numerical solution, while the numerical results obtained with Dymola

Simulation tool	$E(t_f)$	CPU time
MPSSim	$1.13 \cdot 10^{-6}$	0.148s
MapleSim (Rosenbrock)	$0.00 \cdot 10^{-0}$	4.484s
MapleSim (CK45)	$1.50 \cdot 10^{-5}$	1.604s
Dymola	$2.14 \cdot 10^{-3}$	0.890s
OpenModelica	$-1.60 \cdot 10^{-3}$	2.938s

Table 1: Simulation result of the pendulum equation with energy conservation for  $t \in [0s, 100s]$

Simulation tool	$E(t_f)$	CPU time
MPSSim	$1.89 \cdot 10^{-5}$	1.364s
MapleSim (Rosenbrock)	$5.00 \cdot 10^{-6}$	45.358s
MapleSim (CK45)	$1.49 \cdot 10^{-4}$	15.645s
Dymola	$2.14 \cdot 10^{-2}$	8.830s
OpenModelica	—	—

Table 2: Simulation result of the pendulum equation with energy conservation for  $t \in [0s, 1000s]$

are less accurate. The numerical results obtained with MPSSim are accurate within the range of the prescribed error tolerance at low computational costs. However, note that using MPSSim only the costs for the numerical integration of the overdetermined system are measured, while the CPU times of the other tools also contains the costs for index reduction, state selection, projection, and further transformations.

## 6 Conclusions

In this article we have discussed the efficient and robust numerical simulation of dynamical systems that are modeled with MODELICA. We have presented a regularization method for quasi-linear DAEs that is based on an overdetermined system formulation that is obtained by adding all hidden constraints explicitly to the original model equation. The information on the hidden constraints can be obtained from a structural analysis of the system. If a structural analysis cannot be applied these information can be obtained in an analytical way. The overdetermined system formulation can then directly be integrated using a specially adapted numerical integrator. The great advantage of the direct discretization of the overdetermined



formulation is the fact that it is not necessary to determine a selector analytically in advance and that the number of unknowns in the DAE is not increased. A further advantage of an overdetermined regularization with respect to the numerical integration is the possibility to add solution invariants, e.g., mass, impulse or energy conservation laws, to the constraints, which often stabilizes numerical integration. Performing the state selection within the numerical integrator also allows us to switch between different state selections and also opens the door to handle structure varying system models [11]. Currently, no MODELICA simulation framework is able to handle overdetermined system formulations. Therefore, a prototype MODELICA framework MPSSim is presented that includes a translator M2FOR that is used to translate an overdetermined system model provided in MODELICA into FORTRAN source code which can then be integrated using the software package QUALIDAES. MPSSim is still at an early state of development and will be continuously improved.

## Acknowledgements

This work has been supported by the European Research Council through Advanced Grant MODSIM-CONMP.

## References

- [1] R. Altmeyer and A. Steinbrecher. Regularization and numerical simulation of dynamical systems modeled with Modelica. Preprint 29-2013, Institut für Mathematik, TU Berlin, 2013.
- [2] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*, volume 14 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 1996.
- [3] C.W. Gear. Differential-algebraic equation index transformations. *SIAM Journal on Scientific and Statistic Computing*, 9:39–47, 1988.
- [4] E. Griepentrog and R. März. *Differential-Algebraic Equations and Their Numerical Treatment*, volume 88 of *Teubner-Texte zur Mathematik*. BSB B.G.Teubner Verlagsgesellschaft, Leipzig, 1986.
- [5] E. Hairer, C. Lubich, and M. Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Springer-Verlag, Berlin, Germany, 1989.
- [6] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, Germany, 2nd edition, 1996.
- [7] P. Kunkel and V. Mehrmann. Index reduction for differential-algebraic equations by minimal extension. *Zeitschrift für Angewandte Mathematik und Mechanik*, 84(9):579–597, 2004.
- [8] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations. Analysis and Numerical Solution*. EMS Publishing House, Zürich, Switzerland, 2006.
- [9] S. Mattsson and G. Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific and Statistic Computing*, 14:677–692, 1993.
- [10] C.C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistic Computing*, 9:213–231, 1988.
- [11] P. Pepper, A. Mehlhase, Ch. Höger, and L. Scholz. A compositional semantics for Modelica-style variable-structure modeling. In P. Fritzson F.E. Cellier, D. Broman and E.A. Lee, editors, *4th International Workshop on Equation-Based Object-oriented Modeling Languages and Tools (EOOLT 2011)*, number 56 in Linköping Electronic Conference Proceedings, pages 45–54, Zurich, Switzerland, September 2011. September 5, 2011.
- [12] J. Pryce. A simple structural analysis method for DAEs. *BIT Numerical Mathematics*, 41:364–394, 2001.
- [13] L. Scholz and A. Steinbrecher. A combined structural-algebraic approach for the regularization of coupled systems of DAEs. Preprint 30-2013, Institut für Mathematik, TU Berlin, 2013.
- [14] A. Steinbrecher. Analysis of quasi-linear differential-algebraic equations. Preprint 11-2006, Institut für Mathematik, TU Berlin, 2006.

- [15] A. Steinbrecher. *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems*. PhD thesis, Technische Universität Berlin, 2006.
- [16] A. Steinbrecher. Deflating type regularization method for quasi-linear differential-algebraic equations. Preprint, Institut für Mathematik, TU Berlin, 2013. In preparation.

# Symbolic Initialization of Over-determined Higher-index Models

Lennart Ochel    Bernhard Bachmann

lennart.ochel@fh-bielefeld.de    bernhard.bachmann@fh-bielefeld.de

Bielefeld University of Applied Sciences, Department of Mathematics and Engineering  
Am Stadtholz 24 - 33609 Bielefeld, Germany

Francesco Casella

francesco.casella@polimi.it

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria  
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy

## Abstract

The quantity of initial equations required in an object-oriented model can only be determined at system level. Since Modelica models are generally designed by components, it is difficult to calculate the amount of initial equations needed at system level, especially when changes are applied to the model, e.g. by adding or removing components. Therefore, it is more convenient to define initial equations at component level. Due to component connections, algebraic dependencies between states may be introduced, which eventually lead to the removal of states when symbolic index reduction algorithms are applied. In this process, the corresponding initial equations are not automatically removed, which results in an over-determined initial system.

This paper describes an algorithm that detects such redundant equations and determines if they are consistent or not. Consistent redundant initial equations can thus be removed automatically, and inconsistent ones can be reported to the modeler. A prototype of the algorithm is implemented in OpenModelica, tested on several representative cases, and compared to previously presented concepts.

*Keywords: initialization; higher-index; simulation; over-constrained*

## 1 Introduction

### 1.1 Statement of the Problem

Initial equations in Modelica are usually defined at the component level, and they are as many as the dynamic variables of the component, i.e., the potential states.

When connections are made, connection equations can induce algebraic constraints on dynamic variables. The dummy derivatives algorithm is used by many Modelica tools to dispose of some potential states and obtain an index-1 problem. As a result, there will be more initial equations than states, leading to an over-constrained initialization problem.

It is agreed that index reduction is necessary in object-oriented modeling to achieve full modularity without compromises, and suitable means to handle it have been developed over time, so it is obviously necessary to extend the handling to initialization as well. In the majority of cases the over-constrained initialization problem turns out to be consistent, and should therefore be handled automatically, without any intervention by the end user; inconsistent initialization problems should be reported in a user-friendly way.

### 1.2 Overview of Existing Solutions

OpenModelica has been using a numerical approach to solve this problem for a long time, as discussed in [1]. The initialization problem is turned into an optimization problem, where the sum of square of all residuals is minimized; if the problem is consistent, then the minimum is zero, otherwise the inconsistency is spread among equations, which is generally not a good idea. However, solving an optimization problem is much harder and time-consuming than solving a system of equations, and it might easily be possible to get trapped in local minima. Also, convergence problems quickly get worse when increasing the size of the system, up to the point where a solution cannot be reliably found unless guess values very close to the solution are given to all the problem unknowns. This gets even worse in the case of hybrid models, where

some parts of the system contain discrete equations. Even for determined initialization systems, the numerical approach is only applicable in very special cases. In practice, this limits the application of the method to very simple models. So far, large-scale and hybrid models are reliably initialized using the symbolic initialization method described in [2], which has been further developed to also handle over-determined systems.

To our knowledge, there is no other Modelica tool available that has a general approach to handle over-determined initialization problems. The package *ModelicaTest*, which is offered by the Modelica Association together with the Modelica standard library (MSL), has been extended to provide a free-accessible set of appropriate test models. This can be used to compare the excellence of the initialization capabilities of different tools.

### 1.3 Structure of the Paper

In Section 2, an simple introductory example is discussed in detail to demonstrate the problem being tackled in this paper. In Section 3, an algorithm is presented to locate redundant equations which arise when symbolic index reduction is applied, and detect whether they are consistent or not. In Section 4, it is shown how the proposed algorithm works. A few simple test cases and the results of a more involved test case are discussed. Section 5 concludes the work with final remarks and suggestions for future work.

## 2 Introductory Example

This section describes how over-constrained initialization problems arise, by means of a simple electric circuit model, where two series-connected capacitors are connected to a constant voltage source. A graphical representation is shown in Figure 1 and an equation-based model description where all alias variables have been removed is shown in Listing 1.

Both of the capacitors introduce a potential state  $u_i$ . Due to component-based modeling, both capacitors may also introduce initial equations, for example  $u_i = 5$ .

An algebraic constraint among potential states is introduced by connecting the capacitors in parallel to a voltage source, so this model has index-2. Hence, symbolic index reduction is used (see [3], [4]) to transform this system into an equivalent index-1 system of lower order. During this process, one of the potential

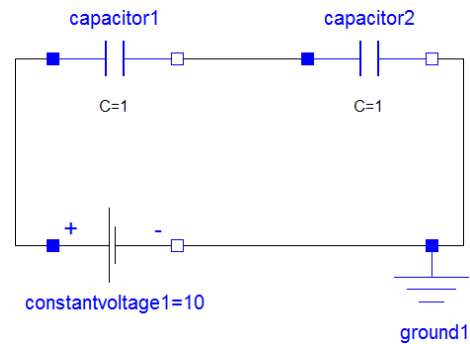


Figure 1: Introductory example - object diagram

states becomes an ordinary algebraic variable and the additional equation  $0 = \text{der}(u_1) + \text{der}(u_2)$  gets introduced to keep the dynamic system determined.

```

1 | model example
2 |   Real u = 10;
3 |   Real u1;
4 |   Real u2;
5 |   Real i;
6 |   parameter Real C = 1;
7 |   initial equation
8 |     u1 = 5;
9 |     u2 = 5;
10 |  equation
11 |    i = C*der(u1);
12 |    i = C*der(u2);
13 |    u = u1 + u2;
14 | end example;

```

Listing 1: Introductory example - flat Modelica model

After the dynamic system is transformed to index-1, the initial equations will be added. As a result, an over-determined system arises that needs to get matched. There will be at least one unmatched equation for each redundant initial equation. Also note that there is no unique matching (besides the matching within each strong component) due to the over-determined system structure; different matchings are possible, which leave out different unmatched equations and possibly lead to different sets of strong components. Figure 2 shows one possible matching that will be used for the next steps; the gray equation is the unmatched one.

If all unmatched equations are consistent, then they can be removed and the initial solution can be calculated using robust and efficient algorithms designed for square problems. In order to check the consistency, the matching digraph gets first transformed into a directed graph by replacing each non-matching edge with an arc going from the E-node to the V-node, then by collapsing each V-node with its matching E-node.

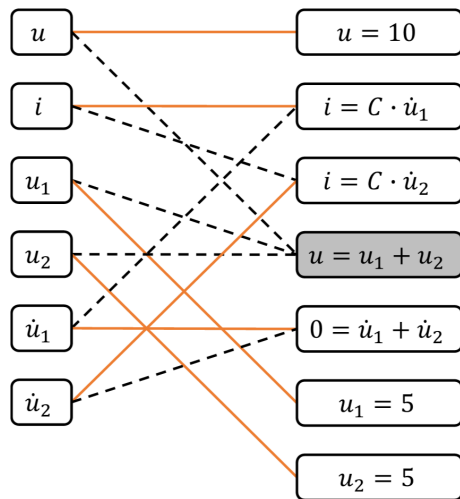


Figure 2: Introductory example - matching

Next, Tarjan’s algorithm [5] gets applied on the directed graph, ignoring the unmatched equations, to find out strong components. The result of this procedure is presented in Figure 3, with one strong component at the top of the graph.

The symbolic consistency check will be performed on this graph. The basic idea is to only consider the sub-graph which involves the initial equations and the unmatched equations, in this case the lower part of Figure 3, and to recursively and symbolically solve the equations starting from the sinks and going up to the unmatched equations. In this case, once the three sink node equations have been solved, the gray equation becomes  $10 = 5 + 5$ , which is equivalent to  $0 = 0$ , and thus redundant.

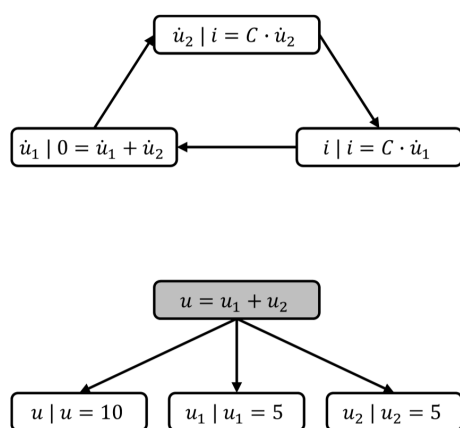


Figure 3: Introductory example - directed graph of the unmatched equation

In general, however, it will not be possible to symbolically solve all the chain of equations leading to the unmatched equation. It is therefore essential to select

a particular matching that makes this possible. A concrete algorithm that does so is presented in the next section.

### 3 Algorithm for Redundant Equation Detection

The goal of this algorithm is to find redundant equations and remove them if they are consistent, or issue an error if they are not. A symbolic approach is preferred, because it avoids the need of setting more or less arbitrary numerical thresholds and using iterative solvers. In general, solving the full initialization problem symbolically is not feasible, because it often contains large and nonlinear coupled systems of equations that cannot be solved in closed form. Therefore, a symbolic approach should aim at finding a suitable subset of the initialization problem that is easy to solve symbolically, to check if there are redundant and consistent equations, so as to remove them.

In a system with  $n$  equations and  $m$  variables, with  $k = n - m$  too many equations, there are  $\binom{n}{k}$  possible sub-sets of equations that may be removed to make the problem square. However, the resulting problem need not only be square, but also have a solution, and many of these sets cannot be removed, because they contain essential (and not redundant) constraints. Hence, an algorithm is needed that efficiently finds those sets that can be removed without losing essential information.

In addition to that, each set of removed equations corresponds to a unique matching (ignoring the different matchings within strong components) of the remaining system. Depending on this matching, a consistence check can be performed by recursively evaluating the sorted subsets. The effort for this evaluation strongly depends on the selected equation dependencies, i.e., on the selected matching.

In practice, such a subset can normally be found, because initial equations are usually linear, involving one unknown, e.g.  $x = x_0$ , or  $\text{der}(x) = 0$ . State derivatives usually show up linearly in balance equations, because they stem from the derivative of some basic quantity (mass, energy, momentum, charge) via differentiation. Connection equations, which usually provide the constraints that make the problem high index, are also linear. In most cases, it should then be possible to symbolically prove that the problem is either consistent or inconsistent, by means of symbolic computations, because the equations to be solved symbolically will belong to the above-mentioned categories.

### 3.1 Proposed Algorithm

1. Apply index reduction and state variable change to the dynamic problem, using the dummy derivatives algorithm.
2. Add the initial equations to the set to form the initialization problem.
3. Build the corresponding E-V graph.
4. Run the matching algorithm; some unmatched equations will remain at the end of the process (one for each redundant initial equation).
5. Transform the E-V digraph into a directed graph by first replacing each non-matching edge with an arc going from the E-node to the V-node, then by collapsing each V-node with its matching E-node.
6. Run Tarjan's algorithm on the directed graph, ignoring the unmatched equations, to find the strongly connected components, and collapse each strong component in a single node.
7. Starting from the sink nodes and proceeding recursively towards the source nodes (unmatched equations), symbolically solve each equation (or system of equations) for its unknown(s) and substitute the result in all the nodes that have arcs pointing to the solved equation node and are needed to validate the unmatched equations.

If the symbolic solution of one equation is not possible (e.g., a sub-expression becomes  $0/0$ ), then try to change the matching from step 4 for the corresponding equation and go back to step 5. If there is no other matching, then the algorithm aborts, and it is neither possible to draw any conclusion on the consistency of the problem, nor to reduce the problem to a square one.

8. If all the unmatched source nodes contain equations equivalent to  $0 = 0$ , then the over-determined system is consistent, and it is possible to turn it into a square equivalent system by just removing all the unmatched equations. If there are one or more source nodes containing equations equivalent to  $0 = 1$ , then the system is inconsistent. For diagnostic purposes, it is possible to report for each node the set of connected equations which are inconsistent. This will help the end user to identify the source of the inconsistency and possibly remove it.

The calculation of the matching in step 4 is essential for this approach. Due to the over-determined system structure, in general various matchings are possible. If a matching for all variables exists that contains no algebraic loops, then it is preferred and should be tried first. Therefore, Tarjan's tearing algorithm described in Cellier's book [6] is applied to the over-determined equation system.

A recursive evaluation of the equation system is possible if and only if during sequential evaluation each next equation depends on exactly one more unknown variable. This means that within the equation system at least one equation exists, which depends just on one variable. Tarjan's algorithm detects this fact, matches the variable to the equation, and reduces the corresponding bipartite graph by removing both nodes and all corresponding edges. The same must hold for the reduced set of equations and can be repeated until all variables are matched.

If during the algorithm all remaining equations depend on at least two unknown variables no matching without algebraic loops exists.

The proposed algorithm extends the existing symbolic initialization method of OpenModelica [2], which is capable to initialize complex hybrid models.

### 3.2 Numeric Fall-back Case

Instead of step 7 and 8 another possible approach would be to leave the unmatched equations out of the problem, solve it, then numerically evaluate the residuals and check if they are small enough. This might be non-trivial in some cases when the involved quantities are very large due to the choice of measurement units and to the size of the system under consideration.

This kind of approach has also the disadvantage that it is not possible to find a different matching if the system ends up in a local singularity.

## 4 Discussion Based on Selected Examples

The proposed algorithm has been tested using the package *OverdeterminedInitialization* from *ModelicaTest*, which has been first created for this purpose at the 80<sup>th</sup> Modelica Design Meeting. This test package contains a list of models from different domains (Electrical, Mechanics, and Fluid) for test purpose, which become over-constrained after index-reduction. The different test cases can be used to cover all kinds of the different issues, that may occur during this process.

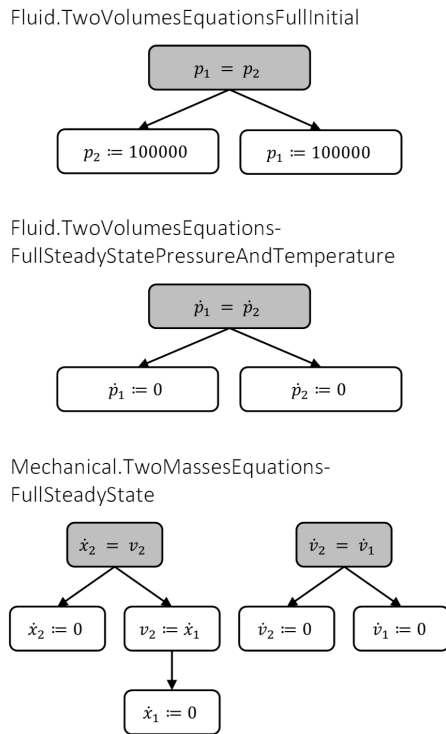


Figure 4: Directed graph of the unmatched equations of some simple examples from the package *OverdeterminedInitialization*

Because of the low complexity of the most test cases, they end up with a similar subsystem for the consistence check as the introductory example. Therefore, the consistence check can be performed by evaluating the remaining subgraphs recursively towards the unmatched equations (see Figure 4).

### 4.1 Fluid Model of Two Volumes

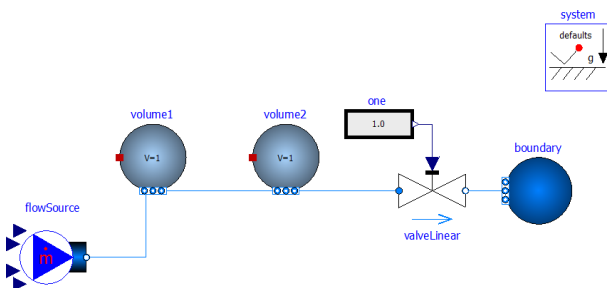


Figure 5: *TwoVolumesEquationsFullSteadyState-MassAndEnergy*

One of the most complex examples of the package is the fluid model *TwoVolumesEquationsFullSteadyStateMassAndEnergy*, in which equations and initial equations refer to stored mass and energy within each volume as differentiated variables, but pressure and

temperature are forced to be states using the Modelica stateSelect attribute. Figure 5 shows a graphical representation using MSL components. Note that, due to the way the Modelica.Fluid components are designed, it is not possible to reproduce this situation, so a textual equation-based model is used for testing the algorithm. This model and the respective dependence graph are listed in the appendix. Using transformations like alias elimination, a reduced version can be generated that contains just 10 variables and 11 equations. Figure 6 shows the reduced dependence graph with one possible matching.

The example contains 11 equations, and is over-constrained due to one equation. Therefore, there are 11 sets of equations (each of cardinality one) that may be removed. Depending on the selected set of equations several cases can occur:

1. recursively evaluable systems
2. systems containing algebraic loops
3. systems with local singularities
4. system with structural singularities

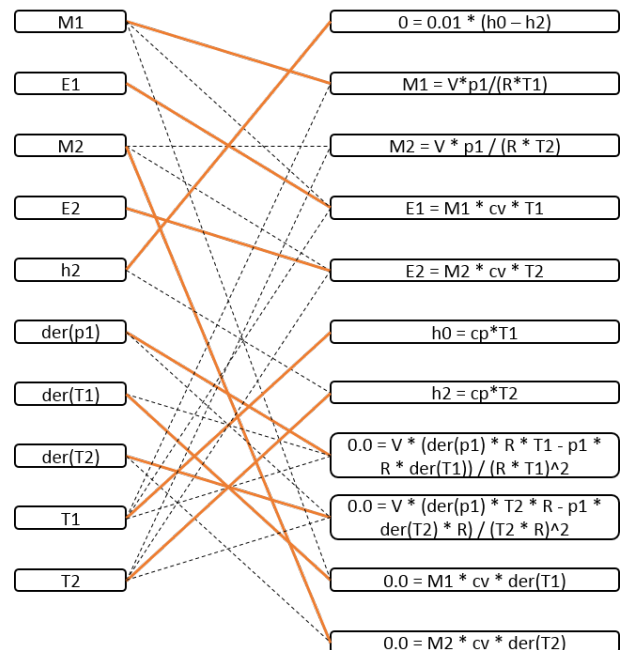


Figure 6: A matching of the remaining equation system of *TwoVolumesEquationsFullSteadyState-MassAndEnergy*

Four sets end up in case 1, which is the most desirable case. Here all equations can be solved recursively towards the unmatched ones to check the consistency

of the system. This happens often in simple cases as described above. Unfortunately, there is no guarantee that such a case will always show up in more complex examples. But, if such a case exists, it will be always captured.

Three sets end up in case 2, which is the most common case for real-world problems. In general, there is no way to avoid algebraic loops. However, this case will only need to be taken into consideration if there is no recursively evaluable system available. In that case, advanced symbolic solvers are needed to compute the solution of these loops symbolically. If this is not possible a modified version of the algorithm could switch to a numeric fall-back mode.

Two sets end up in case 3, which occurs, for instance, if a subexpression is evaluated to 0/0. In this case the selected set of unmatched equations can not be used to determine whether the system is consistent or not. The proposed algorithm detects this in step 7, and, if possible, this case is avoided by rejecting the corresponding set and trying another one, in the hope of finding a recursively evaluable one.

Two sets end up in case 4, which gives a non-valid matching for the system. It is not possible to use the selected set of unmatched equations for any conclusion. Because, the proposed algorithm only selects matchable sets of equations, this case is never reached. Furthermore, due to this fact the possible sub-sets of equations that may be reduced is much less than  $\binom{n}{k}$ .

### 4.2 Electrical 3-Phase System

This test model was already presented in [1]:

Consider the following electrical 3-phase power system of Figure 7, where two generating units VS1 and VS2 are connected via a transmission line modeled by components LR1 and LR2.

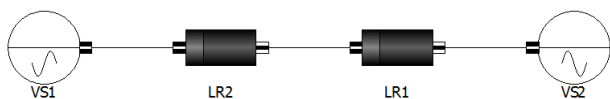


Figure 7: An electrical power system with two generating units VS1 and VS2 connected via a transmission line

The connectors are written in dq0-coordinates implementing the potential variable  $u_{dq0}$  and the flow variable  $i_{dq0}$ . These quantities are constant in case of a non-distributed steady state, which is generally assumed during the initialization process. Introducing the Park-Transformation  $P$  the 3-phase rotating system

(voltages  $u_{abc}$  and currents  $i_{abc}$ ) can be calculated from the dq0-representation and vice versa.

The transmission line (LR1 and LR2) is modeled by a purely inductive and resistive component, based on the Modelica Electrical Library. Since LR1 and LR2 are connected in series, giving a higher index system, index reduction has to be applied for simulation purposes.

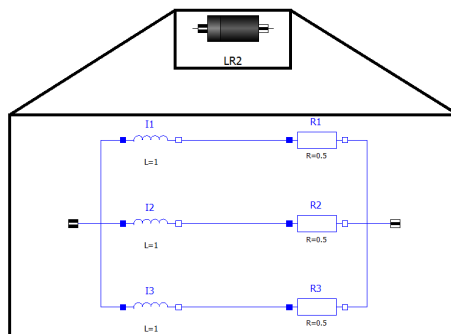


Figure 8: LR2 component with dq0-connectors

The voltage source is described similarly using the Modelica Standard Library combined with the dq0-connectors.

#### 4.2.1 Results Using the Proposed Algorithm

This system covers some important pitfalls. After index reduction, the initial system contains three equations too many. The described approach is not able to find a matching without algebraic loops. Therefore, the resulting loop (with a size of 27) has to be solved symbolically. This is currently not supported from the OpenModelica back-end and might be in general impossible.

As fall-back case the system can be solved numerically during runtime as described in 3.2. For the concrete case of the 3-phase system, the resulting algebraic loop becomes singular if a wrong set of equations is removed. A more advanced solution is described in the following subsection.

#### 4.2.2 Future Work

As stated above, a symbolic solution for the 3-phase system is still needed. One possible idea is to change the consistence check as follows:

Depending on the matching, the following three equations might get selected as the set of removed equations:

$$0.0 = 0.5773502691896258 * (\text{der}(\text{LR2.I1.i}) + \text{der}(\text{LR2.I2.i}) + \text{der}(\text{LR2.I3.i}));$$



$$\begin{aligned}
0.0 = & \text{LR2.Park}[2,1] * \text{der}(\text{LR2.I1.i}) + \\
& \text{LR2.Park}[2,2] * \text{der}(\text{LR2.I2.i}) + \\
& \text{LR2.Park}[2,3] * \text{der}(\text{LR2.I3.i}) + \\
& \text{der}(\text{LR2.Park}[2,1]) * \text{LR1.i\_abc}[1] + \\
& \text{der}(\text{LR2.Park}[2,2]) * \text{LR1.i\_abc}[2] + \\
& \text{der}(\text{LR2.Park}[2,3]) * \text{LR1.i\_abc}[3];
\end{aligned}$$

$$\begin{aligned}
0.0 = & \text{LR2.Park}[1,1] * \text{der}(\text{LR2.I1.i}) + \\
& \text{LR2.Park}[1,2] * \text{der}(\text{LR2.I2.i}) + \\
& \text{LR2.Park}[1,3] * \text{der}(\text{LR2.I3.i}) + \\
& \text{der}(\text{LR2.Park}[1,1]) * \text{LR1.i\_abc}[1] + \\
& \text{der}(\text{LR2.Park}[1,2]) * \text{LR1.i\_abc}[2] + \\
& \text{der}(\text{LR2.Park}[1,3]) * \text{LR1.i\_abc}[3];
\end{aligned}$$

They are quite similar to the next three equations, that are part of the matched system and involved in the algebraic loop:

$$0.0 = 0.5773502691896258 * (\text{der}(\text{LR2.I1.i}) + \text{der}(\text{LR2.I2.i}) + \text{der}(\text{LR2.I3.i}));$$

$$\begin{aligned}
0.0 = & \text{LR1.Park}[2,1] * \text{der}(\text{LR2.I1.i}) + \\
& \text{LR1.Park}[2,2] * \text{der}(\text{LR2.I2.i}) + \\
& \text{LR1.Park}[2,3] * \text{der}(\text{LR2.I3.i}) + \\
& \text{der}(\text{LR1.Park}[2,1]) * \text{LR1.i\_abc}[1] + \\
& \text{der}(\text{LR1.Park}[2,2]) * \text{LR1.i\_abc}[2] + \\
& \text{der}(\text{LR1.Park}[2,3]) * \text{LR1.i\_abc}[3];
\end{aligned}$$

$$\begin{aligned}
0.0 = & \text{LR1.Park}[1,1] * \text{der}(\text{LR2.I1.i}) + \\
& \text{LR1.Park}[1,2] * \text{der}(\text{LR2.I2.i}) + \\
& \text{LR1.Park}[1,3] * \text{der}(\text{LR2.I3.i}) + \\
& \text{der}(\text{LR1.Park}[1,1]) * \text{LR1.i\_abc}[1] + \\
& \text{der}(\text{LR1.Park}[1,2]) * \text{LR1.i\_abc}[2] + \\
& \text{der}(\text{LR1.Park}[1,3]) * \text{LR1.i\_abc}[3];
\end{aligned}$$

Instead of solving the entire algebraic loop symbolically, it might be possible to transform each of the removed equations into an equation of the matched system, and thus prove its redundancy.

The first equation of both sets are already equal, so there is no further consistence check needed. By applying common-sub-expression elimination techniques and advanced alias elimination also the other two equations can be transformed into the other two. For that, it is just needed to figure out that `LR1.Park` is alias of `LR2.Park` and `der(LR1.Park)` is alias of `der(LR2.Park)`.

## 5 Conclusions

This paper has discussed a symbolic algorithm that handles the initialization problem of over-determined systems. The presented approach has not to deal with numerical thresholds and there is no risk of trapping into local minima. Also hybrid models can be handled efficiently. This is a major improvement with respect

to the existing numerical approach within OpenModelica.

This paper focused on symbolic techniques to determine potential redundant equations and ways to analyze whether the system is consistent or not. These work well for problems, which end up in an recursively evaluable initial system. Furthermore, the developed algorithm takes care of singularities, if they occur during the consistency check. More complex problems end up with systems including algebraic loops. If they are not solvable symbolically, a numerical fall-back solution as well as advanced symbolic techniques are proposed.

## References

- [1] B. Bachmann, P. Aronsson, P. Fritzson, "Robust Initialization of Differential-Algebraic Equations", In Proceedings 5th International Modelica Conference, Vienna, Austria, Sep. 4-5, 2006, pp. 607-614
- [2] L. Ochel, B. Bachmann, "Initialization of Equation-Based Hybrid Models within OpenModelica", In EOOLT 2013 Proceedings, Nottingham, UK, April 19, 2013, pp. 97-103
- [3] C.C. Pantelides, The Consistent Initialization of Differential-algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, Vol. 9, No. 2, 1988
- [4] S.E. Mattsson, G. Söderlind, Index Reduction in Differential-algebraic Equations Using Dummy Derivatives. *SIAM Journal on Scientific Computing*, Vol. 14, No. 3, 1993
- [5] R. Tarjan, Depth-first search and linear graph algorithms. *SIAM Journal on Computation*, Vol. 1, No. 2, 1972
- [6] F.E. Cellier, E. Kofman, *Continuous System Simulation*, Springer, 2006

**A Test Model TwoVolumesEquationsFullSteadyStateMassAndEnergy**

```

1  model TwoVolumesEquationsFullSteadyStateMassAndEnergy
2  "Two volumes containing an ideal gas with a zero dp connection"
3  Real M1( stateSelect=StateSelect.avoid , start=1.0),
4  M2( stateSelect=StateSelect.avoid , start=1.0),
5  E1( stateSelect=StateSelect.avoid , start=1.0),
6  E2( stateSelect=StateSelect.avoid , start=1.0),
7  p1( stateSelect=StateSelect.prefer , start=1.0),
8  p2( stateSelect=StateSelect.prefer , start=1.0),
9  T1( stateSelect=StateSelect.prefer , start=1.0),
10 T2( stateSelect=StateSelect.prefer , start=1.0),
11 w0, w1, w2, h1, h2;
12 parameter Real V = 1;
13 parameter Real R = 400;
14 parameter Real cp = 1000;
15 parameter Real cv = cp-R;
16 parameter Real h0 = cp*300;
17 parameter Real Kv = 1e-7;
18 initial equation
19 der(M1) = 0;
20 der(E1) = 0;
21 der(M2) = 0;
22 der(E2) = 0;
23 equation
24 der(M1) = w0 - w1;
25 der(E1) = w0*h0 - w1*h1;
26 der(M2) = w1 - w2;
27 der(E2) = w1*h1 - w2*h2;
28 M1 = V*p1/(R*T1);
29 M2 = V*p2/(R*T2);
30 E1 = M1*cv*T1;
31 E2 = M2*cv*T2;
32 h1 = cp*T1;
33 h2 = cp*T2;
34 w0 = 0.01;
35 w2 = Kv*p2;
36 p1 = p2;
37 end TwoVolumesEquationsFullSteadyStateMassAndEnergy ;

```

Listing 2: Test model *TwoVolumesEquationsFullSteadyStateMassAndEnergy*

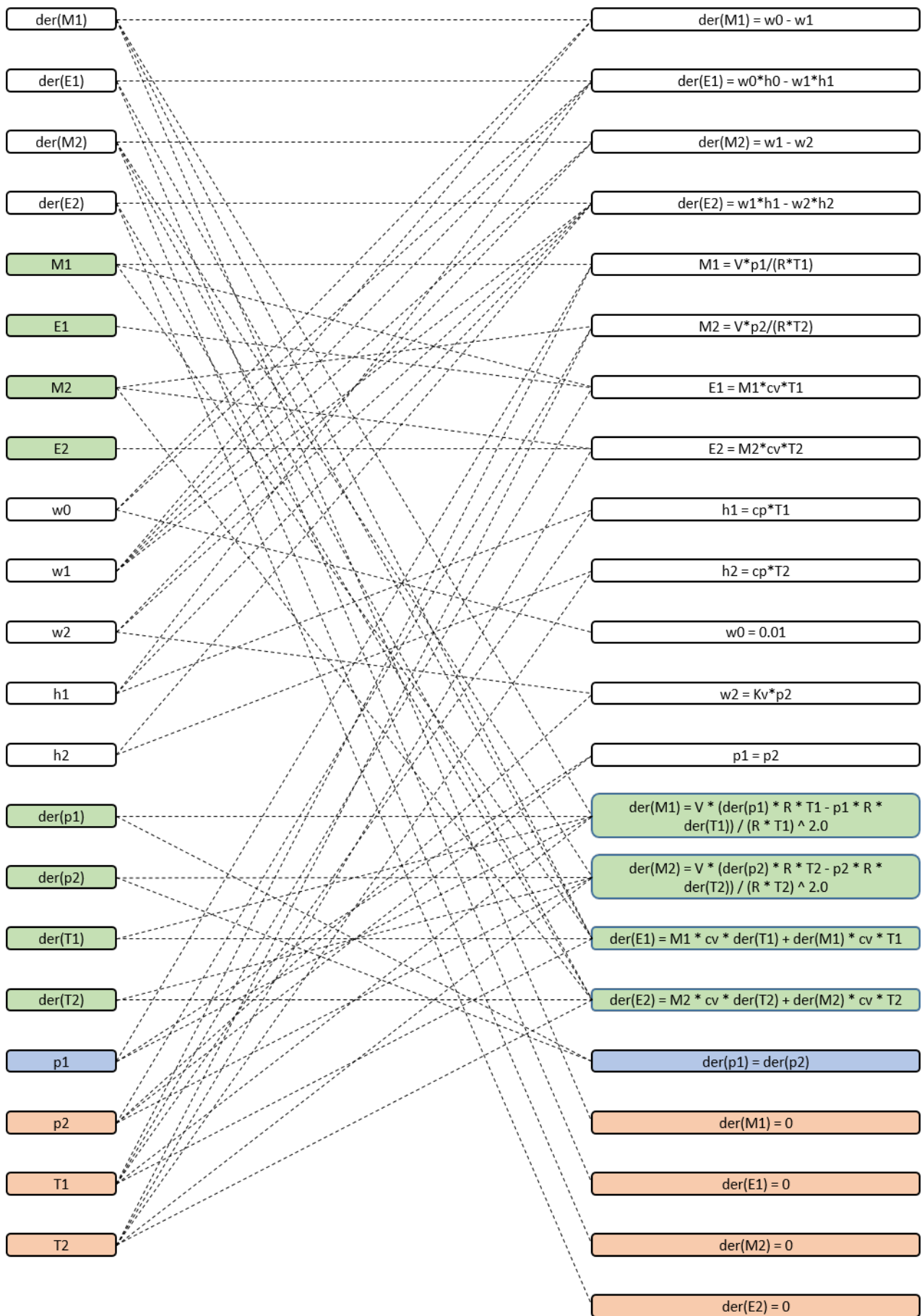


Figure 9: TwoVolumesEquationsFullSteadyStateMassAndEnergy - dependence graph



# Proposal for standardization of Heat Transfer Modelling in NewThermal Library

Susana López      Itzal del Hoyo  
IK4-TEKNIKER

Iñaki Goenaga, 5. 20600 Eibar, Gipuzkoa (Spain)  
[susana.lopez@tekniker.es](mailto:susana.lopez@tekniker.es) [itzal.delhoyo@tekniker.es](mailto:itzal.delhoyo@tekniker.es)

## Abstract

This article presents the `NewThermal` library that extends the capacities of `Thermal` library from the Modelica Standard Library (MSL) including a proposal for standardizing the use of `Material` models. The new library is intended to decouple the models that collect the equations of heat transfer phenomena from the thermo-physical properties of the matters (fluids and solids).

The `NewThermal` library, in the same way that the current `Thermal` library from MSL, is composed of thermal system components to model heat transfer and simple thermo-fluid pipe flow. Nevertheless, the models from the package proposed inherit the thermal properties from `Media` and `Material` models of the fluids and solids involved (either temperature dependent or constant). In this way, the user has three aspects to define; the heat transfer phenomena to be modelled, the geometrical characteristics of the bodies, and the matters involved.

Components inside `HeatTransfer` package are implemented such they can be used for any material model in `Materials` package, in the same way that components from `Modelica.Fluid` were carried out for their use with media models from `Modelica.Media`.

The `NewThermal` library, in addition, provides some general base models for the modelling of 2D and 3D heat conduction in basic solid geometries.

Two examples of use for different domains are presented to illustrate the features of the new libraries.

*Keywords: Modelica; heat transfer; thermal properties; thermal conduction, three dimensional heat flow*

## 1 Introduction

In a general view, any heat transfer phenomena can be described with three different aspects:

- Heat transfer mechanism: conduction, convection or radiation or a combination of them.
- Geometrical characteristics of the phenomena: dimensions of the bodies, relative position between bodies and/or fluids.
- Properties of the substances involved, fluids and solids.

On one hand, in the modelling of any detailed thermal behaviour, the thermal properties of the solids and fluids involved (density, thermal conductivity, specific heat,...) acquire a relevant importance and in many cases it is essential to take into account their dependence on the temperature.

On the other hand, the geometry of real bodies usually can be constructed by the addition of basic geometries, that is, from a macroscopic point of view almost all bodies can be discretized in parallelepipedic, cylindrical and annular nodes (lumped elements) or a combination of them.

Considering this ideas and following the philosophy of replaceable media models in fluid models from MSL[1][2], IK4-TEKNIKER has created the libraries `NewThermal` and `Materials`. This first version provides mainly models for the modelling of conductive heat transfer. But there are described also some guidelines to extend the library to other mechanisms such as convective and radiative heat transfer modelling.

The goal of the `NewThermal` library is to provide standard components for heat transfer modelling dependent on geometry but independent of material properties.

## 2 Materials Library

It is defined as the library of solid material property models. The models from Materials Library are based on the `partialMaterial` partial model. This partial model has declared, as replaceable models, all models required in the modelling of any heat conduction phenomenon. In the figure 1 is shown the replaceable models mentioned above. On one hand, there is the inertia model, required if the heat capacity of the material is taken into account, and on the other hand the models needed for the modelling of heat transport inside the material are available. The latter are chosen depending on the geometries involved, illustrated in the figure 2.

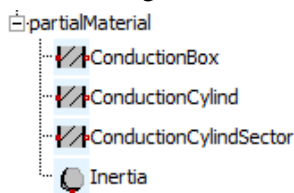


Figure 1: Components of `partialMaterial` class

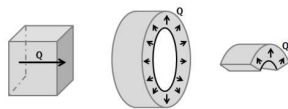


Figure 2: Box, cylindrical and cylindrical sector geometries

Hence, material models extend from this partial model and add in each case the thermal properties of the substances to be modelled.

That is, in the same way that models from `Modelica.Fluid` library with models from `Media` library, any thermal model can include a replaceable instance of `partialMaterial` which allows selecting models from Materials library choosing between a large list of materials to inherit their thermal properties, as it is shown in the figure 3.

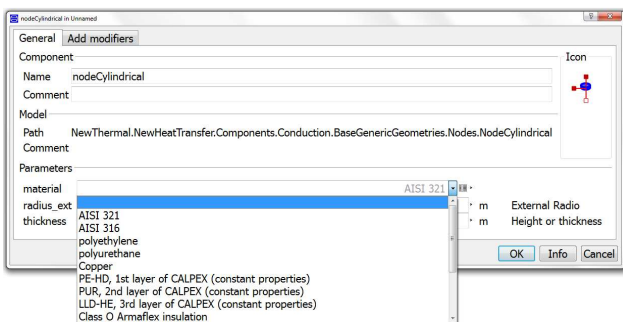


Figure 3: Component for the modelling of the thermal behaviour of a body with cylindrical geometry.

In this case, the external radius and the height of the cylinder are parameters and the specific heat capacity, density as well as conductivity of the material extends from the replaceable material model.

Current `partialMaterial` model includes only replaceable models for the modelling of conductive heat transfer but it is possible to extend it in the future adding replaceable models relative to radiative heat transfer.

## 3 NewThermal Library

### 3.1 General Library Structure

The `NewThermal` library has the same structure as `Thermal` library from `MSL`, as it is shown in the figure 4. All new models are built with connections on existing `Interfaces` package, so they are compatible with any model inside `Modelica.Thermal`.

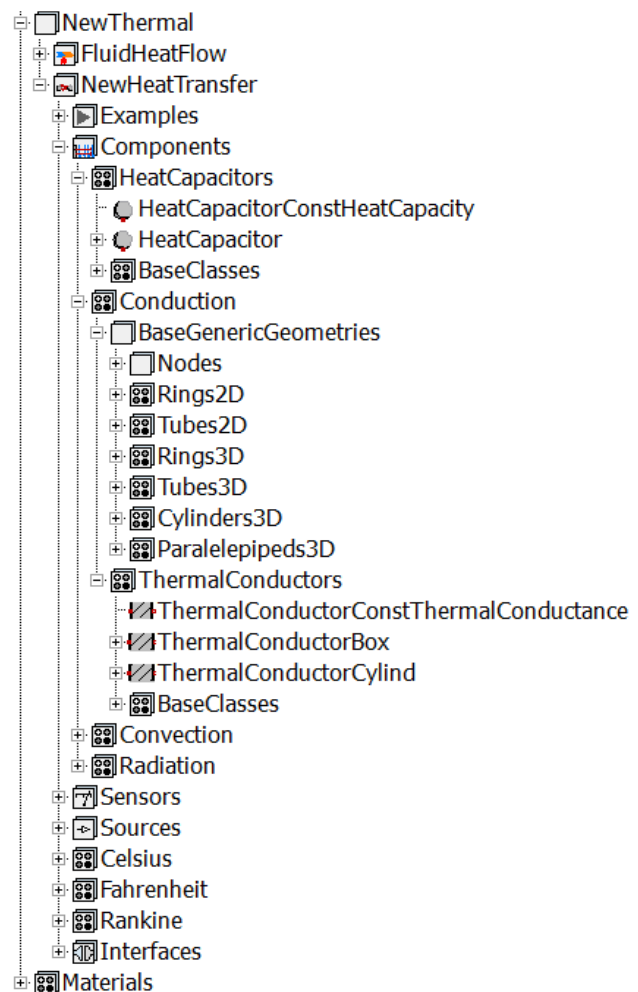


Figure 4: `NewThermal` library structure

Most of new models are based on the use of replaceable material models and for this reason it is essential to open it together with Materials library.

Following sections are focused on HeatCapacitors and Conduction packages that support simulation of heat conduction through any solid.

### 3.2 HeatCapacitor Library

As shown in the library structure, HeatCapacitor subpackage is included in Components package. In this subpackage, a collection of generic models for the heat capacity of a material can be found. In these components, no specific geometry is assumed beyond a total volume with uniform temperature for the entire volume. Furthermore, only one parameter it is required, precisely the volume of the body to be modelled, in such a way that the heat capacity value is inherited from the replaceable material model. The figure 5 shows the information expected from the user when the heat capacitor model is used.

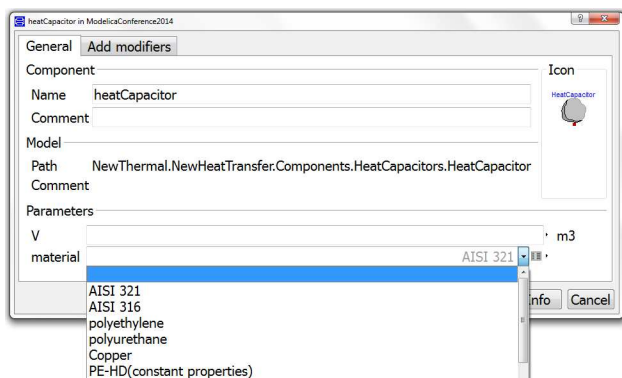


Figure 5: heatCapacitor class

### 3.3 Conduction Library

The Conduction subpackage from NewHeatTransfer package is composed of two groups of models.

In the first one, named ThermalConductors, a collection of thermal conductor models can be found, for 1D conductive heat transport. It contains a sort of models for different geometries with the option to choose the material of the body to be modelled.

The second library, called BasicGenericGeometries, collects a large list of models for the simulation of conduction heat transfer in any 2D or 3D geometry (parallelepipeds, cylinders, rings or tubes) and again all of them have the option to choose the material to be modelled.

#### 3.3.1 Heat Conduction Base Classes

The base classes of the ThermalConductors package are partial models that define the interface and the equation of the heat conduction phenomena. Models of ThermalConductors have only geometry values as parameters, inheriting the value of conductivity from the replaceable material model. The figure 6 shows the information to be defined by the user to model heat conduction in box type geometry.

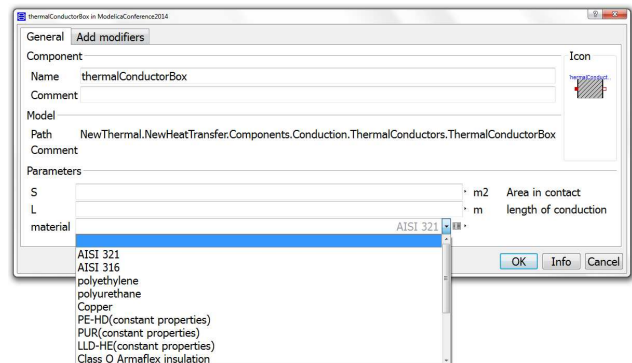


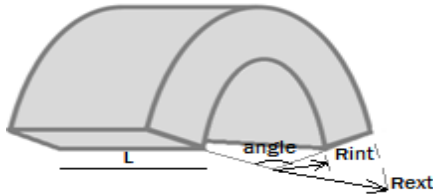
Figure 6: thermalConductorBox model for the transport of heat through box geometry in any material

partialThermalConductor provides the thermal conduction equation  $Q_{flow} = G \cdot dT$ , where  $G$  is defined as thermal conductance of material and its calculation depends on the geometry through which the heat flows. The partial model extends from existing Element1D which provides two thermal ports port\_a and port\_b which are common in all components for modelling heat transfer, such as heat convection or heat radiation.

ThermalConductors package offers a set of models which extend from partialThermalConductor partial model adding the equation characteristic of  $G$  depending on geometry:

- $G = \frac{k \cdot A}{L}$  for a box geometry under the assumption that heat flows along the box length.
- $G = \frac{2 \cdot \pi \cdot k \cdot L}{\log\left(\frac{R_{ext}}{R_{int}}\right)}$  for a cylindrical geometry, under the assumption that heat flows from the inside to the outside radius of the cylinder.

- $$G = \frac{angle * k * L}{\log\left(\frac{R_{ext}}{R_{int}}\right)}$$
 for a sector for any cylindrical geometry, under the assumption that heat flows from the inside to the outside radius of the cylinder.



In all the equations k is the thermal conductivity of the material.

### 3.3.2 Nodes Base Classes for different 2D and 3D geometries

BaseGenericGeometries package in Components.Conduction contains a set of components for the modelling of 2D and 3D heat conduction through bodies with different geometries. These models are built on arrays of components from Nodes package, a collection of models of basic units (lumped elements) for the construction of geometries above mentioned. This approach has been suggested in previous works such as [3].

Thereby, in Nodes package, it can be found annular nodes, parallelepipeds nodes and cylindrical nodes. In the figure 7 the model and icon for parallelepiped node in 3D are shown.

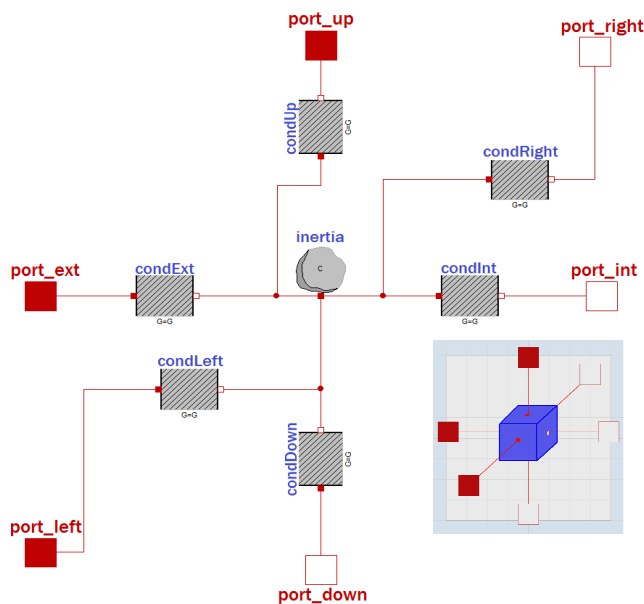


Figure 7: Basic 3D node class

All these components are composed of:

- A heatCapacitor model, from Heat-Capacitor library, for the thermal storage capacity of the node and it is assumed a uniform temperature in it.
- Four/six thermalConductor models, from ThermalConductors library, for modelling the transport of heat. Each couple of thermalConductor models describes the heat transport along the two/three dimensions.

### 3.4 Convection and Radiation Libraries

The models from Convection library, with two heat ports as it is common in all components for modelling heat transfer, are prepared for connecting one of the heat ports to the solid and the other heat port to the fluid, in such a way that the component itself, has as internal variables the temperature at solid and fluid and the heat flow rate from/to solid and fluid. Following the same philosophy used for the creation of HeatCapacitor and Conduction libraries and taking advance of the accessible fluid and solid temperatures involved, all components in Convection library have been provided by a replaceable medium model and correlations for natural and external forced convection [4][5] (forced convection problem inside conduits is already solved in Modelica.Fluid). Thereby, the user only has to provide the geometry information of the body involved in the convective heat transfer and to select the media.

Models in Radiation library respond to the  $Q_{flow} = Gr * \sigma * (port\_a.T^4 - port\_b.T^4)$  radiation formula, where Gr is function of the emittance of material surfaces involved in the thermal radiation and its calculation depends on the geometry and arrangement of surfaces. The emittance of surfaces tends to vary its value with the temperature and this effect is more relevant as the temperature increases [6]. The emittance is considered to be included in material models because it is an intrinsic property of material surface. Therefore, the same as in Conduction library, the Radiation package would offer models for the modelling of thermal radiation depending on the geometry and the arrangement of objects involved but independent on materials.



## 4 Examples

### 4.1 Insulated Pipe

Based on the `dynamicPipe` class from `Modelica.Fluid` and `NewThermal` features, a model of an insulated pipe has been created. Although the model has been named `insulatedPipe`, it describes the hydraulic and thermal behaviour of any pipe with one or more solid layer(s) assuming radial symmetry in all phenomena. Hence, with this model the user can model from a simple copper tube of any home heating system to a more sophisticated plastic tube with some insulation layers.

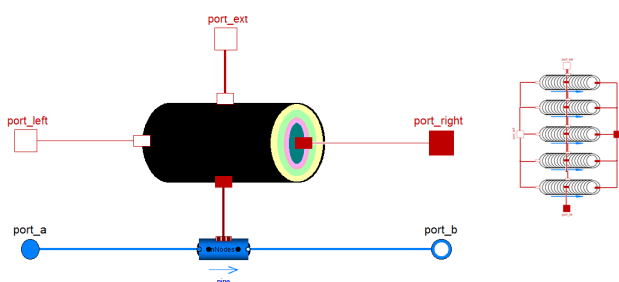
The cover of the dynamic pipe is compounded at most of five layers of any material from `Materials` library. The first layer is the closest to the fluid, that is, the layer with the smallest radius.

The user has to indicate how many layers the pipe has and the thickness of each layer, as well as the layers materials and the internal radius of the first layer.

If the number of layers is less than five, the user can disable the layers no required, switching to false the corresponding boolean `useLayer`. On the left side of the figure 8, it is shown the two systems of the insulated pipe:

- the hydraulic system defined by the `dynamicPipe` model from `MSL`
- the thermal system for the cover of the pipe, defined by the `multipleCylinder` model

The right side of the same figure shows the five layers of the cover, each one can be disabled if it is not required for the modelling of the insulated pipe.



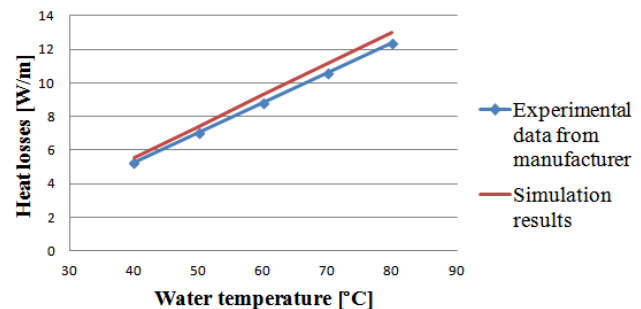
**Figure 8: `insulatedPipe` (left) and its `multipleCylinder2D` base class (right)**

The `insulatedPipe` model has been built easily with several instances of the 2D tube class from `BaseGenericGeometries` package in `NewThermal.HeatTransfer.Conduction`

(right side of the figure 8). It is prepared to be divided into `nNodes` equally spaced segments along the flow path, therefore, the same number of lumped elements are in all matter layers, as much to fluid line as to cover layers.

The model, in addition, has the option of neglecting the axial conductive heat transfer (parallel to the fluid path) throughout the solid materials of the pipe. This assumption is quite reasonable in some cases, depending on the thermal properties of constituent materials, and it can significantly speed up simulations.

The `insulatedPipe` model has been used for the modelling of pre-insulated pipes in a small District Heating system [7] with good results, error of roughly 5%, in comparison with technical report of heat losses provided by the pipe manufacturer.

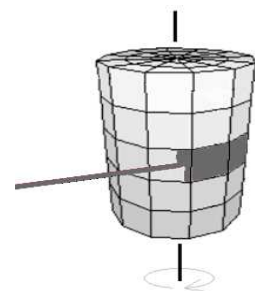


**Figure 9: Experimental data and simulation results for `insulatedPipe` model adapted for commercial pipe simulation.**

### 4.2 Heat transfer in 3D cylinder

Based on models from `Tubes3D` library a discretized cylinder was built. The model was used for simulating laser surface hardening process of a crankshaft. In this case, a 3D model was absolutely indispensable because there was a relative movement between the crankshaft and the punctual heat source (punctual laser).

In this case, `Modelica` model was especially interesting in order to define and also simulate the control strategy of the process.



**Figure 10: Simplified sketch of the interaction between the punctual heat source and the crankshaft**

Before using it for this purpose, 3D tube model was validated with a FEM model (MSC NASTRAN®). Both models were discretized in the same way and exposed to the same external conditions to compare the evolution of the temperature in some nodes.

The tube was discretized in:

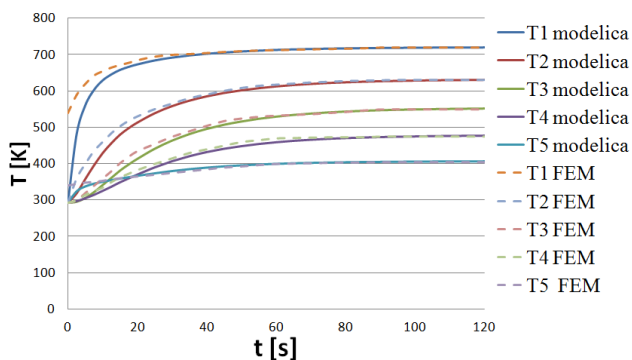
- Number of rings (radial direction): 20
- Number of sectors: 16
- Number of slices (axial direction): 5

The following external conditions were assumed:

- Convection in internal and external lateral surfaces:
  - $h_{\text{internal wall}}$ : 10 W/m<sup>2</sup>K
  - $h_{\text{external wall}}$ : 100 W/m<sup>2</sup>K
- Constant temperature in the upper and lower external surfaces:
  - Upper surface: 500 °C
  - Lower surface: 100 °C
- Temperature of nodes at  $t=0s$ , 20°C

The good agreement between stationary simulation results of both models can be appreciated in the figure 11.

The discrepancy on transient behaviour is due to different initial conditions in some nodes. All the nodes in the FEM model could not be initialized at 20°C due principally to the way of imposing external conditions. In this case, the nodes from the upper slice and the lower slice were in contact with an imaginary plate at 100°C and 500°C, so that, this nodes inherited directly the temperature of them at  $t=0$ . In Modelica model, nevertheless, all nodes started the simulation at 20°C.



**Figure 11. Temperatures of five nodes of the FEM and Modelica models**

## 5 Conclusions

NewThermal together with Materials libraries have been created by IK4-TEKNIKER in order to extend MSL capabilities for heat transfer modelling. The new libraries decouple material properties from heat transfer phenomena modelling and allow taking into account the influence of temperature on material properties such as thermal conductivity, specific heat capacity, etc.

NewThermal library provides also basic models to simulate 2D and 3D heat conduction problems in bodies with simple geometries.

Two practical examples from different application domains have been shown to demonstrate the use of the libraries.

## References

- [1] Modelica Association, (2012). A Unified Object-Oriented Language for Physical System Modeling, Modelica®
- [2] Franke R., Casella F, et all. Standardization of Thermo-Fluid Modeling in Modelica.Fluid. Como, Italy, 7th Modelica Conference
- [3] Tiller M. Introduction to Physical Modeling with Modelica. Kluwe Academic Publisher
- [4] Chapman A., Fundamentals of Heat Transfer. Macmillan
- [5] Incorpera F.P., DeWitt D.P. Fundamentals of Heat Transfer, John Wiley & Sons Inc
- [6] Howell J.R., Siegel R., Pinar Mengüç. Thermal Radiation Heat Transfer. CRC Press Taylor & Francis Group
- [7] ASHRAE Handbook: Heating, Ventilating & Air-Conditioning Systems; 2008

# A Modelica Power System Component Library for Model Validation and Parameter Identification

Luigi Vanfretti<sup>1,2</sup> Tetiana Bogodorova<sup>1</sup> Maxime Baudette<sup>1</sup>

1: Smart Transmission Systems Lab. (SmarTS Lab), Electric Power Systems Department  
KTH Royal Institute of Technology, Stockholm, Sweden.

2: R&D Division, Statnett SF, Oslo, Norway

## Abstract

This paper summarizes the work performed in one of the work-packages of the FP7 *iTesla* project. This work consisted in the development of a power system component library for phasor time domain simulation in Modelica.

The models were used to build power system network models, used in experiments for parameter identification. The experiments were carried out with the RAPID toolbox, which has been developed at SmarTS Lab within the same project. The toolbox was written in MATLAB, making use of FMI Technologies for interacting with Modelica models.

*Keywords:* Power Systems, Phasor Simulation, Modelica, FMI, Parameter Identification, Model Validation.

## 1 Introduction

### 1.1 Conventional Power System Modeling and Simulation

Modeling power system components and networks is important for different studies in the planning and operation of electricity networks [1], and the level of complexity in the models depends on the type of studies carried out [2]. Traditional tools for power system modeling are usually tied to a specific time scale, often limiting the applicability and/or validity of the models to a specific kind of studies [3].

The models can be of Electro-Magnetic Transient (EMT) type [4, 5], which is the most detailed type of power system models; of phasor time-domain (phasor) type, which use a simplified representation commonly used in positive sequence phasor time-domain simulation for stability assessment [2]; or could even use a Quasi Steady State (QSS) representation for simplified long term dynamic simulation [6]. To assemble power

system models using different components of specific type and perform simulations, domain-specific tools are traditionally used; for example [7, 8] are used in the case of phasor-time domain simulation.

Other modeling and simulation approaches which are less utilized in the domain is the dynamic phasors approach [9], and mixed EMT & phasor simulation [10, 11].

The reminder of this article focuses on phasor time-domain power system modeling using the Modelica language and the application of Flexible Mockup Interface (FMI) Technologies for model validation and parameter identification.

### 1.2 Motivation — Modelica, an unambiguous modeling language for Power Systems

Several tools for phasor time domain simulation already exist, but only few of them let the user access the models' code (mainly those that are open source software [12]), lack flexibility in regards to modeling and simulation features provided [2], and have limitations for unambiguous model exchange [13]. This motivated the work presented in this paper which is part of the FP7 *iTesla* project [14], where a power system component library of phasor type models is being developed.

Modelica was chosen as the modeling language for this library as it allows to develop models using a formal mathematical description and because it separates the model from the solver [3]. The models developed in Modelica use an explicit mathematical representation waving out any ambiguity about the model, while enabling further seamless simulation with diverse tools [3].

Modelica is an object-oriented programming language where the parameters of a model are the object's attribute. This facilitates the development of power

system models composed by instances of components previously developed.

To the knowledge of the authors, there have been two previous efforts in the use of Modelica for power system modeling and simulation: the ObjectStab [15] library, and the SPOT [16] library which evolved into the commercial “Electric Power Library” by Modelon AB [17]. While these libraries could have been used for the purposes of this article, there are social aspects of resistance to change [18] that prompted the authors to develop a new library. For example, users of a specific power systems tool are skeptical about other tools different from the one that they use — this is a complex behavior [19] that involves both the change in the human interactions that accompany the use of a specific tool, as well as a concern about the technical aspects of a new technology.

To make a transition where the use of Modelica is accepted by power system practitioners, the authors and other colleagues have used different strategies to overcome the resistance of domain experts. This approach is similar to that in [20]: a first strategy is to promote the use of Modelica in power systems as a medium for unambiguous model exchange [3] which is an Alpha [20] strategy that supports the common goal of model exchange; a second strategy (type Omega [20]) was used to decrease avoidance forces: made a software-to-software validation of each power system component of *trusted* domain-specific power system tools where simulation results between the Modelica library in different tools are appraised against a domain-specific tool [21].

Further details on power system modeling using Modelica as approached in this article are available in [21, 3].

### 1.3 Exploiting Modelica and FMI Technologies

Modelica models can also be exploited through Functional Mockup Interface (FMI) [22]; a standard for model exchange between different tools supporting and implementing the standard. As such, Dymola [23], OpenModelica [24] and JModelica.org [25] implement the standard for both import and export; as well as co-simulation.

FMI Technologies offer wide possibilities for model re-use within different software tools. Thus, the models developed and used in Modelica are not tied to the development environment, contrarily to traditional tools for power system modeling. This flexibility

was taken advantage of to develop a Rapid Parameter Identification toolbox (RAPID).

## 1.4 Paper Organization

The remainder of this paper is organized as follows. The power system component library is described in Section 2. Details of two of the models developed by SmarTS Lab and their validation are shown in Section 3 and a model validation experiment using the RAPID toolbox is presented in Section 4. Conclusions are drawn in Section 5

## 2 Library Structure

The library was developed for power systems phasor time-domain simulation in Modelica. It is comprised by different packages that build its main structure, which is described next.

### 2.1 Structure

A total of 120 models are categorized into the following upper level packages (see Fig. 1):

- Connectors (2):  
The package is comprised by two connectors, one, PwPin, is adopted for treating voltage and current as complex variables. The other, ImPin is a simple connector for real variables.
- Electrical (56):  
The main package of the library with all the power system component models for phasor time domain representation. It is divided into several sub-packages.
- Examples (17):  
A set of examples using the power system component models.
- Interfaces (2):  
Blocks for adding some inputs and outputs to the models.
- NonElectrical (45):  
A set of functions extended from the Modelica standard library and adapted with the connectors of this power system components library.

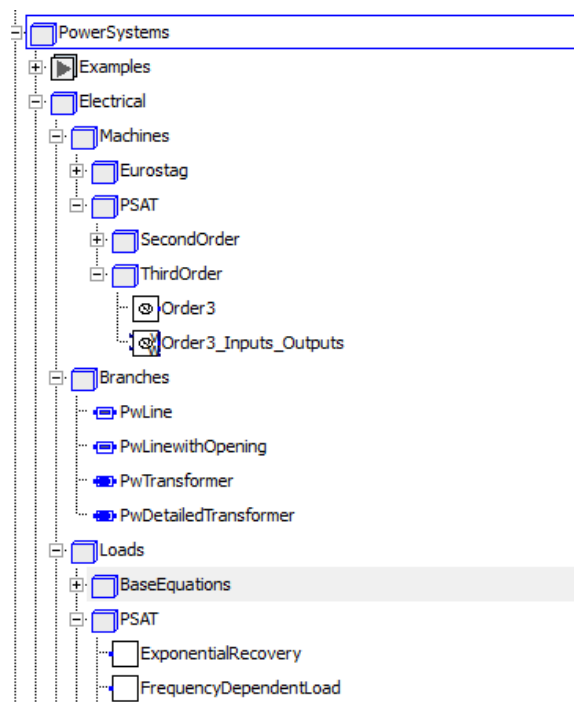


Figure 1. Partial screenshot of the library

## 2.2 Electrical Components

As a joint effort from the different project participants, the power system components models have mostly been transcribed from the *Eurostag* [8] and PSAT [26] software.

Different components such as transmission lines, transformers, buses and some events (line opening and faults) were developed as generic models. All load models, were implemented following the representation used by PSAT and *Eurostag*. Generator models have also been implemented, including detailed models as used in *Eurostag* as well as different models from PSAT. Additionally some non traditional generators have been modeled including photo-voltaic models (3) and two wind turbines models. These models enable the simulation of complete power system networks with any kind of perturbation.

The models developed at SmarTS Lab are the following:

- Load models from PSAT (Exponential recovery, Frequency dependent, Constant PQ, Voltage dependent, ZIP, ZIP Jimma and Mixed loads)
- An Automatic Voltage Regulator (AVR) from PSAT (Type III)
- A Turbine Governor (Type II) from PSAT
- Synchronous generators (second and third order) from PSAT

- Solar photo-voltaic panel (two simplified models from PSAT and detailed model from KTH)
- Doubly Fed Induction Generator (DFIG) wind turbine (PSAT model and GE model)

These models have been developed together with examples, which served for parameter identification experiments such as the one presented in Section 4.

## 3 Implementation and Validation of Power System Component Models

All models developed at SmarTS Lab have been created from a reference model taken in another domain specific simulation environment. They have been subject of a Software-to-Software validation against the reference model by comparing simulation results.

In this Section, two of the models developed at SmarTS Lab are presented in details together with their validation results.

### 3.1 GE Wind Turbine Model

This first study case illustrates the development of a wind turbine generator model taking as reference a MATLAB/SIMULINK implementation of the wind turbine-generator model for grid studies used by General Electric Energy (GE). The wind turbine-generator is Type III, which commonly refers to an induction generator with the rotor winding connected to the grid through a back-to-back converter. The descriptive reference provided by GE Energy can be found in [27].

#### 3.1.1 Modeling

The reference was implemented with function blocks in MATLAB/SIMULINK, organized in three different subsystems (Turbine, Electrical Control and Generator). The system was developed with the voltages (real and imaginary parts) as inputs and currents (real and imaginary parts) as outputs. The system also has an input for the wind speed.

For the Modelica model, the model structure and the block diagram were preserved. The main motivation for this approach was to maintain the existing initialization algorithm used in the reference model. This algorithm is responsible for initialization of, among others, all the integrators in the model.

The resulting model is shown in Fig. 2, where the top-level layer of the model can be observed. The major difference is the integration of the PwPin connector to interface the model to other components, such as

transmission lines. This connector contains two standard variables used for the real and imaginary part of the voltage, as well as two flow variables used for the real and imaginary part of the current.

The Modelica model also implements the initialization algorithm aforementioned. This implementation required to adapt the code to the Modelica syntax by replacing all for-loops by while-loops, as in Modelica for-loops cannot be exited before completion of all iterations.

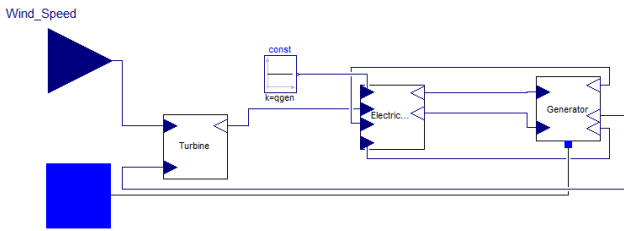


Figure 2. Modelica implementation of the GE Wind Turbine-Generator Model

The original wind turbine reference model was integrated within a test system comprised by a network model, an infinite bus and a wind generator. The network model was built using an admittance matrix, which in the Modelica implementation was replaced by several transmission lines in series. The remainder of the model in Modelica was implemented with components available in the library. A wind generator was additionally developed in Modelica, to create different wind profiles. The resulting test model is presented in Fig. 3.

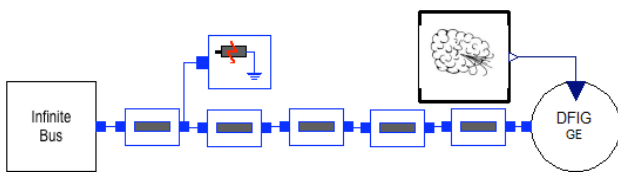


Figure 3. GE Wind Turbine Modelica Test Model

### 3.1.2 Validation results

Figure 3 presents the test model, which includes the simulation scenario comprised by a gust of wind at  $t = 5$  s and a fault at one transmission line at  $t = 10$  s.

Both models have the same simulation scenario and are simulated in their respective environment. However, the solver used is different in each environment, a fixed time-step solver ode3 was selected for the reference model, whereas an adaptive time-step solver DASSL was selected for the Modelica model.

A comparison between the simulation of the two models is presented in Fig. 4, where it can be observed that the simulations produce the same response. This validates the implementation carried out in Modelica.

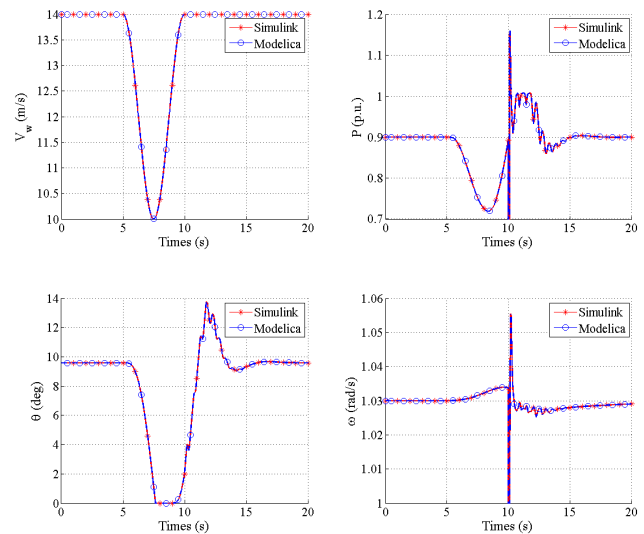


Figure 4. Comparison Modelica vs Simulink for the GE Wind Turbine-Generator Model

However, some differences can be noticed when inspecting the responses more closely. Figure 5 presents an example of this observation. The differences are of a very small order and can be explained by the accuracy of the solvers used in each simulation environment.

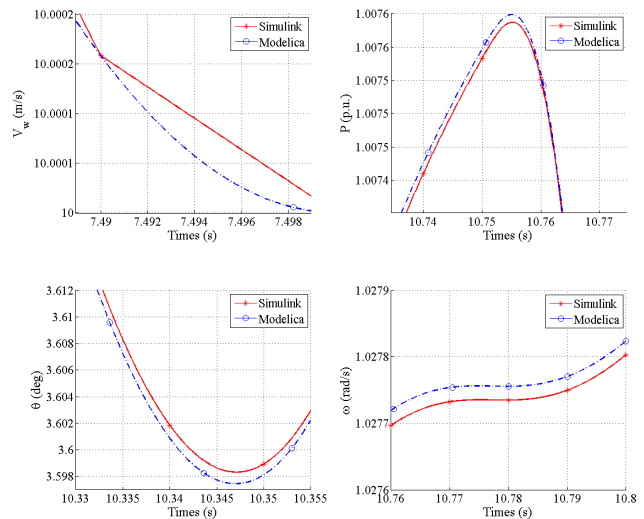


Figure 5. Zoom on comparison Modelica vs Simulink for the GE Wind Turbine-Generator Model

## 3.2 Hydro Turbine Governors

The second study case illustrates the development of a hydro turbine and governor model. The reference

is a PSAT implementation of type 5 hydro turbine and governor (HTG Type 5) modeled at KTH and included into PSAT. The HTG Type 5 models includes a PI controller combined with servomotor (turbine governor model) and a nonlinear turbine model, its Modelica implementation is shown in Fig. 6.

The hydro turbine and its governor are widely used for control of mechanical power delivered by the generator, and often show complex dynamic behavior. Due to this fact, it is a perfect component to show the advantages of using the Modelica language instead of conventional power system modeling tools for simulation purposes. In this section two Software-to-Software validation cases are presented. The first shows the benefits of using Modelica with respect to PSAT. The second presents an alternative validation approach where the validation of the HTG Type 5 model is carried out in a hybrid SIMULINK-Modelica model exploiting the FMI standard which allows to include an FMU as a part of the original MATLAB/SIMULINK system.

### 3.2.1 Modeling

The reference model was implemented in MATLAB/SIMULINK using the built-in Control library for the HTG Type 5 and the SimPowerSystems (SPS) library for the single-machine-infinite bus (SMIB) system. A similar system (with a finite load) was modeled in PSAT and taken as a reference for Software-to-Software validation. Next, the Modelica model of the HTG Type 5 and the same system as in PSAT were built in Dymola (Fig. 7).

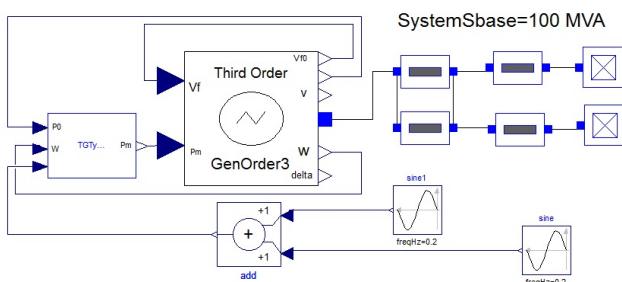


Figure 7. Modelica model of Single machine governed by HTG Type 5 with perturbation

In this the first case, the same initial values are used in Modelica and PSAT, this allows us to carry out the Software-to-Software validation not only in terms of the modeling approach, but also using the same initial conditions.

The types of connectors used in the Modelica model were described above. All the blocks which were

used for the HTG model can be found in the standard Modelica library. For building the hybrid SIMULINK-Modelica model, an FMU file was generated using Dymola to allow simulation in MATLAB/SIMULINK-SPS (Fig. 8). The model was encapsulated into one block, with ability to set up input parameters for the block (Fig. 9) using the FMI Toolbox [28].

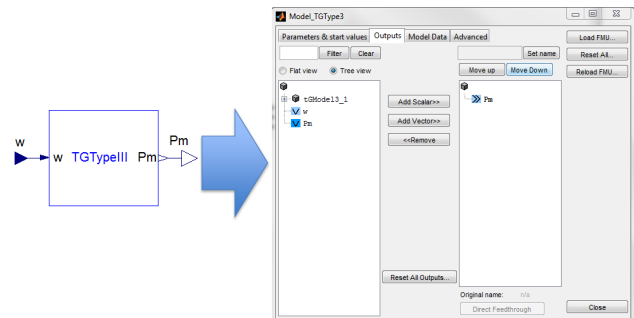


Figure 8. FMU from the Modelica model imported into the FMI Toolbox

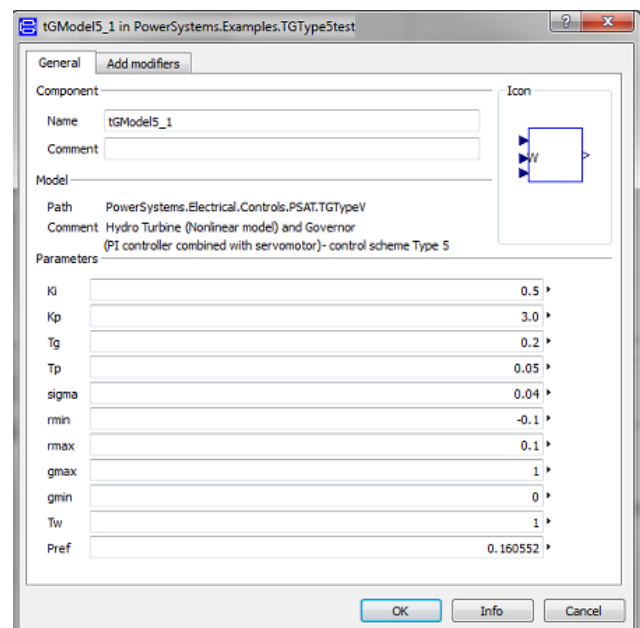


Figure 9. Parameters of HTG Type 5 block

Thus, the HTG Modelica model was used to replace the turbine and governor SIMULINK block in the power system model. This allowed simultaneous simulation and comparison of the response between the SPS models and the Modelica models within the same simulation tool (Fig. 10).

### 3.2.2 Validation results

For validation of the hydro turbine and its governor the input (speed) and output (mechanical power) were compared (Fig.11). The results show that the “shape”

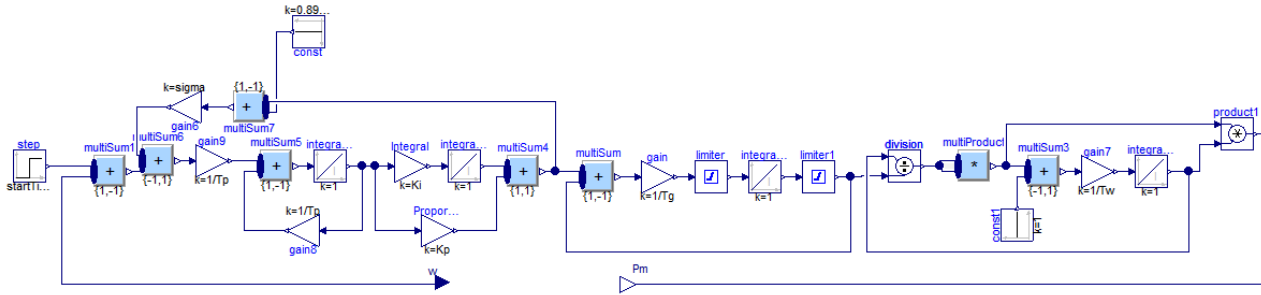


Figure 6. Modelica model of HTG Type 5

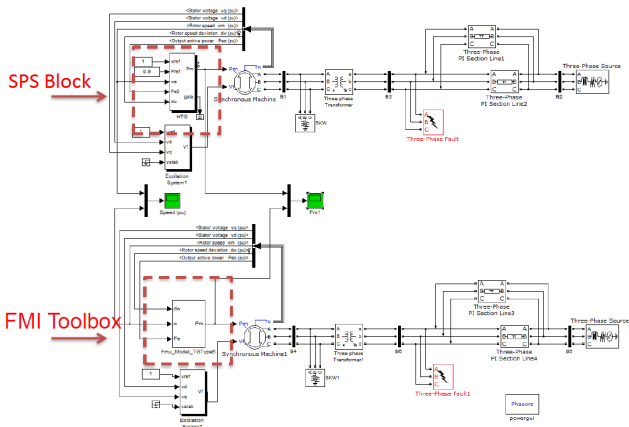


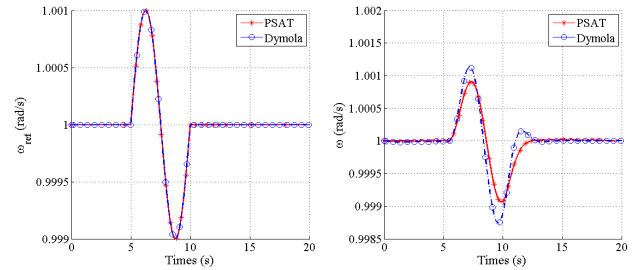
Figure 10. Validation of the HTG model encapsulated using the FMI Toolbox for MATLAB

of the signals match well, but there are some differences. The solver used for simulation with Dymola was DASSL. The differences are due to limitations on the numerical accuracy of the solver in PSAT (a trapezoidal integration with fixed time-step). Please observe that PSAT has a limited number of solvers (trapezoidal integration and forward Euler’s method) and that the time-domain simulation methodology follows domain-specific practices not common to general purpose tools.

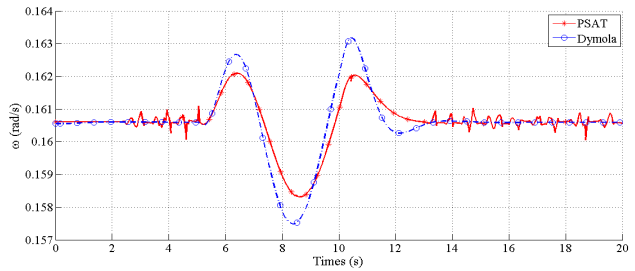
In second validation method, which uses SIMULINK models with the Modelica model embedded in an FMU block and the MATLAB/SIMULINK-SPS reference model, the speed and the output mechanical power were compared (Fig. 12). The results show a very good match between the corresponding outputs.

### 4 Model Validation using Modelica and FMI Technologies

This power system component library has been developed within the FP7 *iTesla* project with the goal of using the models for power system model validation. The choice of Modelica as modeling language



(a) The input (speed) of the HTG Type 5



(b) The output (mechanical power) of the HTG Type 5

Figure 11. Comparison Modelica vs PSAT for the HTG Type 5’s responses with a perturbation at the speed reference  $w_{ref}$ .

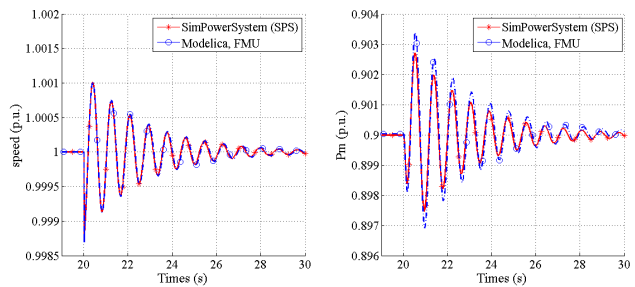


Figure 12. Validation of the HTG model encapsulated using the FMI Toolbox for MATLAB

for this work was also motivated for its support of the FMI standard, which enabled the development of the RAPID toolbox.

#### 4.1 FMI Technologies

The Functional Mock-up Interface (FMI) standard was created as a tool independent format for model ex-



change and co-simulation. The full specifications of the standard, with a list of the supporting tools, are available in [22]. By opening the possibility of using a model in disparate simulation environments, the standard allows to benefit from functionalities offered by different programs.

As another part of the FP7 *iTesla* project, a parameter identification MATLAB toolbox was also developed. This toolbox was built with FMI technologies and uses the Modelica models presented above. The power system models used are test models including an device to be calibrated against some reference signals. An example of using the toolbox for calibrating the parameters of a generator model is presented next.

### 4.2 Example

This parameter identification example computes seven of the generator model parameters using reference measurement from the simulation of an EMT model (4 outputs: the voltage magnitude, the rotor speed, and the active and reactive powers). The reference model in this case was built in MATLAB/SIMULINK-SPS. The authors attempted to substitute the complex dynamics of the EMT generator model with a simplified third order generator model. The other components of the system were assumed to be known and modeled with the same components. For this experiment the FMI Toolbox from Modelon AB [28] has been exploited. The reasons for that are the following. First, MATLAB was chosen for the development of the RAPID toolbox. Second, the authors decided to use the Modelica language for power system modeling purposes [3]. The Modelica model with two different test scenarios was constructed (Fig. 13). In both test scenarios the reference signals are perturbed, the nominal torque value in the first and the field voltage in the second.

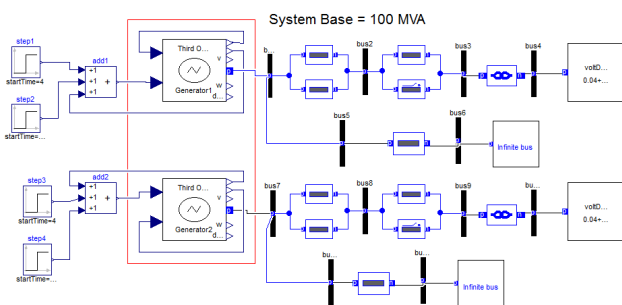


Figure 13. Modelica model used in the parameter identification process (component to be identified bounded in red)

The algorithm of the identification process is the following:

lowing:

1. Collect measurement data from the reference SIMULINK model.
2. Create the power system model to be identified in Modelica. The Modelica model requires a power flow solution, in this case the results of the power flow were obtained from PSAT.
3. Compile an FMU from the Modelica model in Dymola.
4. Create a SIMULINK model using the FMU block from the FMI Toolbox for MATLAB, see Fig. 14. The only blocks required in order to carry out the simulation are: *FMUme* and *To Workspace*. All data inputs and the scopes are included only to monitor the process interactively.
5. Start the RAPID Toolbox.
6. Provide RAPID with appropriate settings (algorithm, parameters, variables, etc.).

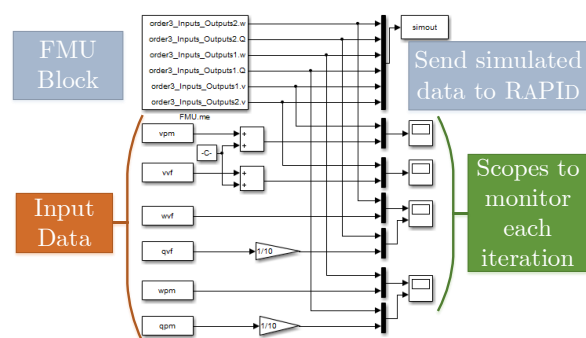


Figure 14. SIMULINK model used by RAPID with the FMU of the Modelica model

### 4.3 Results

The results of identification experience are presented in Fig. 15 and Table 1. The values of the identified parameters are shown in Table 1.

The numerical values for all the parameters were constrained before running the optimization algorithm to a valid range of real valued numbers, typical of synchronous machines. Thus, the resulting parameters are within practical and realistic values. Figure 15 shows the graphical comparison between the simulations in MATLAB/SIMULINK-SPS and Modelica. It

Table 1. Generator parameter estimation results

Parameter	Value
Armature resistance ( $R_a$ )	0.0010156
Direct axis reactance ( $X_d$ )	4.2924
Direct axis transient reactance ( $X'_d$ )	1.37
Direct axis transient time const. ( $T'_d$ )	2.6156
Quadrature axis reactance ( $X_q$ )	5.3994
Inertia coefficient ( $M$ )	14.9005
Damping ratio ( $D$ )	0.0088415

shows that the responses match with an acceptable error. The key here is to remember that a reference signal (from a high order model (MATLAB/SIMULINK-SPS)) is matched to a low-order model. The differences in reactive power (Fig. 15(a)) and in the voltage magnitude (Fig. 15(b)) are acceptable if one takes into account the difference in complexity of the mathematical presentation of high order dynamics by the simplified model. In order to validate the identification results, the simulations in both MATLAB/SIMULINK-SPS and Modelica were repeated using perturbations two-times larger than the original experiments used for identification. Similar results as those reported above were obtained.

## 5 Conclusion

This article introduced the work performed by SmarTS Lab and other collaborators within the FP7 *iTesla* project for developing a library in Modelica for power system component models. This library is already quite versatile as it enabled to carry out model calibration experiments within work package three of the FP7 *iTesla* project.

But the library is still not complete, additional models are being developed at KTH SmarTS Lab to continue populating it. To this aim, the method for developing and validating new models described in Section 3 is used. As such, the newly developed models can be integrated in the library after a successful Software-to-Software validation step.

The overall motivation for the development of such library in the Modelica language was briefly presented in Sections 1 and 4. The potential of model exchange across different tools is indeed of great importance. This is especially true for future power system tools [13], which will require more advanced simulation methods that the currently available in domain-specific tools. For example, unambiguous model exchange will allow for the development of model-driven

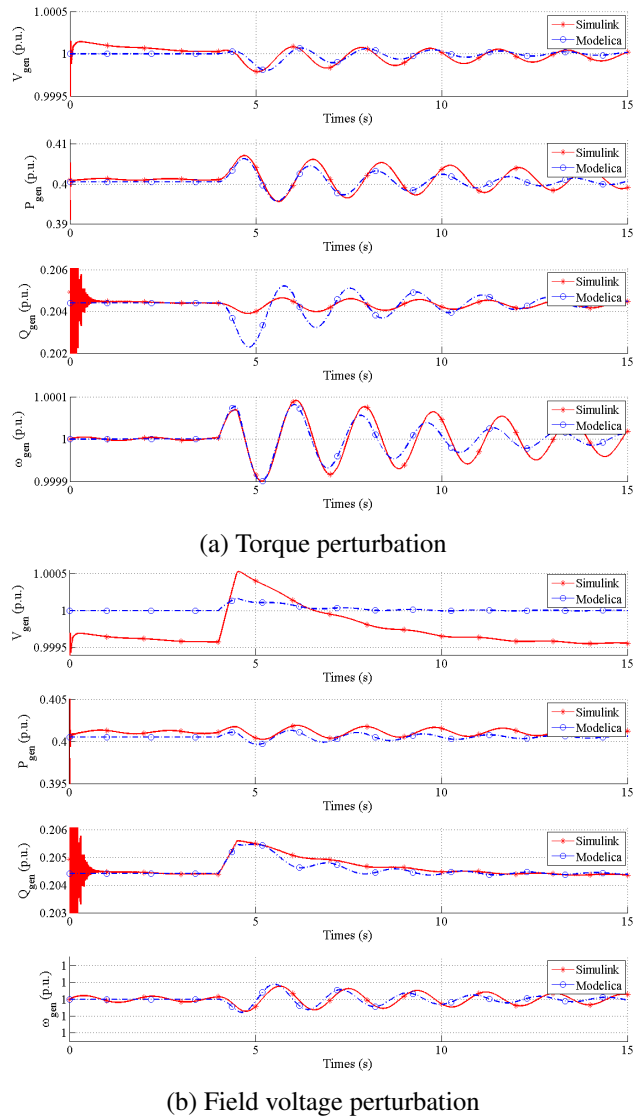


Figure 15. Comparison between the reference (Simulink) and the identified (Modelica) model responses with a perturbation at  $t = 4sec$ .

design of wide-area controls, which will require the use of a cyber-physical modeling language and appropriate solvers.

## Acknowledgement

The support Dr. Juan Sanchez-Gasca at GE Energy in providing a MATLAB/SIMULINK model of a wind turbine-generator is gratefully acknowledged.

L. Vanfretti was supported by Statnett SF, the Norwegian Transmission System Operator, the STANDUP for Energy collaboration initiative, the EU funded FP7 *iTesla* project and Nordic Energy Research through the STRONG<sup>2</sup>rid project.

T. Bogodorova was supported by the EU funded FP7 *iTesla* project.

M. Baudette was supported by the EU funded FP7 *iTesla* project and by Statnett SF, the Norwegian TSO.

## References

- [1] M. Butts and H. S. Smith, "Dynamic oscillations predicted by computer studies," *IEEE Computer Applications in Power*, vol. 4, no. 1, pp. 47–51, 1991.
- [2] F. Milano, *Power System Modelling and Scripting*. Springer, 2010.
- [3] L. Vanfretti, W. Li, T. Bogodorova, and P. Panciatici, "Unambiguous Power System Dynamic Modeling and Simulation using Modelica Tools," *IEEE PES General Meeting*, 2013.
- [4] J. Mahseredjian, V. Dinavahi, and J. Martinez, "Simulation Tools for Electromagnetic Transients in Power Systems: Overview and Challenges," *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1657–1669, 2009.
- [5] H. Dommel, "Digital computer solution of electromagnetic transients in single-and multiphase networks," *Power Apparatus and Systems, IEEE Transactions on*, vol. PAS-88, no. 4, pp. 388–399, 1969.
- [6] T. Van Cutsem, Y. Jacquemart, J.-N. Marquet, and P. Pruvot, "A comprehensive analysis of mid-term voltage stability," *IEEE Transactions on Power Systems*, vol. 10, no. 3, pp. 1173–1182, 1995.
- [7] *PSS/E Program Operation Manual*, Power Technologies, Inc., Dec. 1996.
- [8] Eurostag. Accessed: 2013-12-04. [Online]. Available: <http://www.eurostag.be>
- [9] S. Sanders, J. Noworolski, X. Liu, and G. C. Verghese, "Generalized averaging method for power conversion circuits," *Power Electronics, IEEE Transactions on*, vol. 6, no. 2, pp. 251–259, 1991.
- [10] V. Jalili-Marandi, F. Ayres, C. Dufour, and J. Belanger, "Real-time electromagnetic and transient stability simulations for active distribution networks," in *International Conference on Power System Transients (IPST)*, 2013.
- [11] F. Plumier, C. Geuzaine, and T. V. Cutsem, "A multi-rate approach to combine electromagnetic transients and fundamental-frequency simulation," in *International Conference on Power System Transients (IPST)*, 2013.
- [12] F. Milano and L. Vanfretti, "State of the art and future of oss for power systems," in *Power Energy Society General Meeting, 2009. PES '09. IEEE*, 2009, pp. 1–7.
- [13] E. Lambert, X. Yang, and X. Legrand, "Is CIM suitable for deriving a portable data format for simulation tools?" *IEEE PES General Meeting*, pp. 1–9, July 2011.
- [14] *iTesla: Innovative Tools for Electrical System Security within Large Areas*. [Online]. Available: <http://www.itesla-project.eu/>
- [15] I. Navarro, M. Larsson, and G. Olsson, "Object-oriented modeling and simulation of power systems using Modelica," in *IEEE Power Engineering Society Winter Meeting*, vol. 1, 2000, pp. 790–795.
- [16] SPOT - Free library providing components to model power systems both in transient and steady-state mode. [Online]. Available: <https://github.com/modelica-3rdparty/SPOT>
- [17] Modelon AB. Electric power library description. [Online]. Available: <http://www.modelon.com/products/modelica-libraries/electrical-power-library/>
- [18] J. Ford, L. Ford, and A. D'Amelio, "Resistance to change: The rest of the story," *Academy of Management Review*, vol. 33, no. 2, pp. 362–377, 2008. [Online]. Available: <http://amr.aom.org/content/33/2/362.abstract>
- [19] P. Lawrence, "How to Deal with Resistance to Change (HBR Classic)," *Harvard Business Review*, Jan. 1969.
- [20] J. Kavanagh, "Resistance as Motivation for Innovation: Open Source Software," *Communications of the Association for Information Systems*, vol. 13, no. 36, 2013.
- [21] T. Bogodorova, M. Sabate, G. León, L. Vanfretti, M. Halat, J. Heyberger, and P. Panciatici, "A Modelica Power System Library for Phasor Time-domain Simulation," in *IEEE PES Innovative Smart Grid Technologies (ISGT)*, october 2013.
- [22] Functional Mock-up Interface. Accessed: 2013-12-04. [Online]. Available: <https://fmi-standard.org/>
- [23] Dymola, Dassault Systems. [Online]. Available: <http://www.dymola.com>
- [24] OpenModelica, Open Source Modelica Consortium (OSMC). Accessed: 2013-12-04. [Online]. Available: <https://www.openmodelica.org/>
- [25] JModelica.org - an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems, Moledon AB. [Online]. Available: <http://www.jmodelica.org/>
- [26] PSAT: Power System Analysis Toolbox. [Online]. Available: <http://faraday1.ucd.ie/psat.html>
- [27] K. Clark, N. W. Miller, and J. J. Sanchez-Gasca. Modeling of GE Wind Turbine-Generators for Grid Studies. Version 4.5, Accessed: 2013-12-04. [Online]. Available: <http://goo.gl/x8NBvc>
- [28] FMI Toolbox for Matlab, Modelon AB. Accessed: 2013-12-04. [Online]. Available: <http://www.modelon.com/products/fmi-toolbox-for-matlab/>



# Modelica Model for the youBot Manipulator

Rhama Dwiputra Alexey Zakharov Roustiam Chakirov Erwin Prassler  
Bonn-Rhein-Sieg University of Applied Sciences, Department of Computer Science  
Grantham-Allee 20, 53757 Sankt Augustin

## Abstract

This paper presents the development of Modelica model for the youBot manipulator. Whereas other robotic simulations focus on the robot interaction with its environment, Modelica allows the modeling of the manipulator controllers and motors. The model was developed with a Modelica library for the manipulator's components which provides modularity, reusability and abstraction. A comparison test with the actual system is performed to ensure the model accuracy. The test result shows promising result and provides possible future work. The Modelica model of the youBot manipulator is freely available.

*Keywords:* Control; Manipulator; Modelica; youBot

## 1 Introduction

Models and simulation tools are crucial in robotic research. Although there have been major improvements in the electronic and mechanical field, robots are still expensive equipments. The use of models and simulation tools overcome this problem. Models and simulation tools allow researchers and university students to experiment with different robots. Furthermore, experimentation with models is cost-efficient and time-efficient due to its ability to be automated, conditioned and accelerated.

The *youBot* is a mobile manipulator designed to serve as the reference platform for industry, research and education [1]. Due to its frequent use as a test subject for educational purpose or investigation of new methods in research institute, a model of the youBot is highly advantageous. Robotic simulation tools which has a model of the youBot are VREP [2], We-bots [3] and Gazebo [4]. Like most robotic simulation software, these software focus on simulating the robot interaction with its surrounding environment (navigation, object manipulation) and have its limitation when simulating the robot's internal components (mechanical, electrical, and control system).

Modeling the robot's internal component requires multi-domain capability such as provided by the Modelica<sup>1</sup> description language. Modelica is a non-proprietary, object-oriented, description language for multi domain modeling. Modelica is maintained by the non-profit Modelica association. As such, Modelica is suitable for use in education and research. The work in this paper is influenced by the existing manipulator model in the Modelica Standard Library or MSL [5].

The youBot standard configuration consists of an omnidirectional mobile platform and a five DOFs manipulator with a two finger gripper. In this paper, the manipulator model is developed by dividing the system into several smaller components. The component models are stored in a new Modelica library and categorized in different packages based on its functionality. This approach enables the user to experiment with the manipulator model on the component level.

A model is a representation of the actual system and the benefit of having a model only holds true when the model is accurate. Simulation can result in wrong conclusion when the researcher forget the limitations and condition under which the simulation is valid [6]. Therefore, the development of the manipulator model is followed by a test with the actual system. The test compares the behaviors of the actual system and the model throughout a point-to-point motion. The model accuracy along with the influence of estimated values and approximation is analyzed in the comparison test.

This paper is organized as follows. After this introductory section, Modelica related robotic research is presented in Section 2. Section 3 presents the specification of the youBot manipulator and Section 4 describes the Modelica Library for the youBot manipulator. Afterward, section 5 presents the evaluation of the developed model. Finally, section 6 summarizes the work and provides possible future work.

---

<sup>1</sup>[www.modelica.org](http://www.modelica.org)

## 2 State of The Art

Modelica has been used for modeling spider robotic arm [7], 6-axis industrial robots [8, 9, 10], 3 DOFs parallel Gantry-Tau robot [11], 5 DOFs manipulator [12] and mobile platforms [13, 14]. In most cases, a robot model in Modelica is used for investigating the manipulator's motion control especially in the domain of optimization and system dynamics. Such research requires the repetition of motions and adjustments to the controller which can have damaging effect when being executed on a real robot.

[9] performed optimization through iteration to find a compromise between acceleration, velocity and energy consumption and [10] solved the minimum time optimization problem for an industrial robot. [8] derives the inverse dynamic model of a manipulator using algorithms for differential-algebraic equation available in the Dymola<sup>1</sup> software. Dymola was also used in [12] to design a picking manipulator for agriculture purposes. [11] develops method for kinematic calibration with the Modelica model of parallel Gantry-Tau robot. Aside in the field of motion control, Modelica robot models have also been used for tele-manipulation [7], robot communication [14] and teaching tools [13].

As shown from the work presented in this section, there is a wide range of research with robot models in Modelica. The Modelica model of the youBot manipulator will enable such research to be performed. Since the youBot is designed to be the reference platform for academic institute, a Modelica model of the youBot manipulator is of high importance.

## 3 The youBot Manipulator

The specification of the manipulator is acquired from the following sources:

- official youBot website<sup>2</sup>,
- email communication with the official distributor of the youBot<sup>3</sup> and
- discussion with researchers from BRICS<sup>4</sup> who were involved in the development of the youBot's software.

<sup>1</sup>[www.3ds.com/products-services/catia/portfolio/dymola](http://www.3ds.com/products-services/catia/portfolio/dymola)

<sup>2</sup><http://youbot-store.com/>

<sup>3</sup>[info@locomotec.com](mailto:info@locomotec.com)

<sup>4</sup><http://www.best-of-robotics.org/>

This section consist of two subsections, *kinematic chain* and *control system*. Due to the nature of the robot which is actively being developed, the description presented is subject to changes.

### 3.1 Kinematic chain

The youBot manipulator is a serial chain manipulator with five revolute joints (shown in Figure 1). The manipulator is equipped with a two-finger gripper as its end-effector and each finger weights 0.01 kg. The fingers' body, position and motion has insignificant influence to the system dynamic when compared to the overall manipulator system. Therefore, the gripper is modeled only for the visualization purpose.

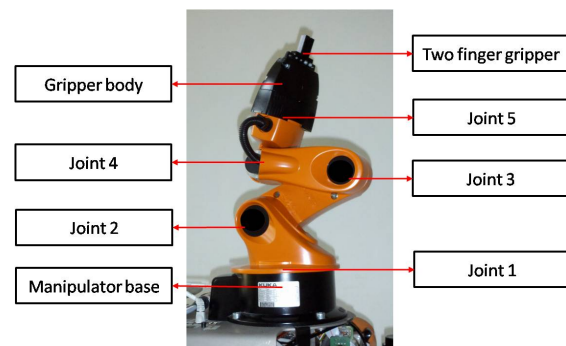


Figure 1: The youBot manipulator

The manipulator is 65.5 cm high when fully extended, weights 6.3 kg and has a payload of 0.5 kg. Each joint is actuated by brushless DC motors and gearboxes with different specifications. The kinematic chain, joint ranges and dynamic properties of the manipulator are presented in appendix A.

### 3.2 Control System

The control system accommodates position, velocity and current control in each joint. For each joint, the control system consists of: 1. three cascaded *proportional-integral-derivative* or PID controllers, 2. a velocity ramp or v-ramp generator and 3. a space vector pulse width modulation (SVPWM). Two modes are available for joint position control, *PID* and *v-ramp* mode. The PID mode calculates the joint velocity in a PID controller whereas in the v-ramp mode, a trapezoidal velocity profile will be generated by the v-ramp generator for the joint velocity. In this paper, the developed model is based on the joint position control in PID mode. Figure 2 shows the overview of the manipulator's controller.

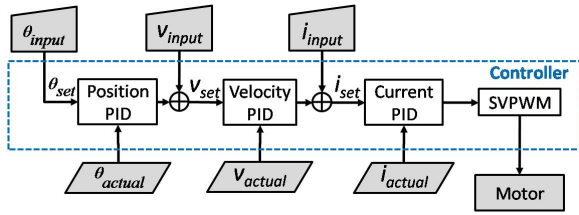


Figure 2: Controller overview

Where  $\theta$  is the joint angle,  $v$  is the joint velocity and  $i$  is the motor current. The *set* variables ( $\theta_{set}$ ,  $v_{set}$ ,  $i_{set}$ ) are the input values for the PID, the *actual* variables are the values from the manipulator's sensors and the *input* variables are the user defined values. When controlling the joint position, a user provides the  $\theta_{input}$  for the controller and the *Velocity PID* receive the output of the *Position PID* as its  $v_{set}$ . When controlling the joint velocity, a user provide the  $v_{input}$  for the controller which is directly forwarded as  $v_{set}$  to the *Velocity PID* (the output of the *Position PID* in such cases will be ignored). The *Position PID* is replaced with the v-ramp generator in v-ramp mode. The PID controllers for position, velocity and current have similar architecture. As a representative of the PID controllers, Figure 3 shows the overview of the PID controller for velocity (*Velocity PID*).

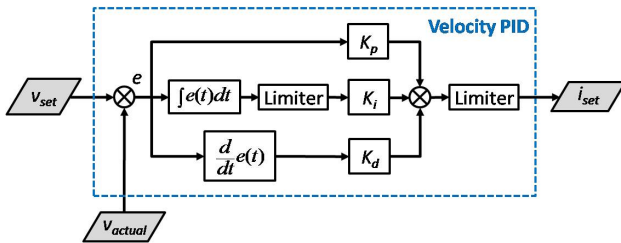


Figure 3: Velocity PID overview

Where  $e$  is the difference between the set value and the actual value.  $K_p$ ,  $K_i$ , and  $K_d$  are the gain parameters for the controllers. The output of the *Velocity PID* is forwarded to the *Current PID* as  $i_{set}$ . As observed in Figure 3, the *Velocity PID* controller is similar to the text book PID as follows:

$$C = K_p e(t) + K_i \int_{t-\Delta t}^t e(t) dt + K_d \frac{d}{dt} e(t) \quad (1)$$

Where  $C$  is the controller output and  $\Delta t$  is the PID period. However, the gain parameter in the velocity PID adjusts itself based on the motor velocity as follows:

$$k = \begin{cases} k_2 & \text{if } |v| \geq a \\ k_1 + \left(\frac{|v|}{a} * (k_2 - k_1)\right) & \text{if } |v| < a \end{cases} \quad (2)$$

Where  $k$  is the gain parameters ( $K_p$ ,  $K_i$  or  $K_d$  in Equation 1),  $k_1$  is the lower boundary of the gain parameter, and  $k_2$  is the upper boundary of the gain parameter value,  $v$  is the motor velocity and  $a$  is the threshold value for the motor velocity. The *Position PID* has the same characteristic as the *Velocity PID*. Therefore, the *Position PID* and the *Velocity PID* are referred as the non-linear PID. The non-linear PID enables the user to set different control behaviors for low and high velocity. Similar to the gain parameters, limiters in the position and velocity controller have non-linear characteristic where the limit value is defined by the motor velocity.

## 4 The youBot Modelica Library

The Modelica library for the youBot manipulator in this paper is developed with Dymola. The library is developed using a “divide and conquer” principle with emphasize on modularity, re-usability and abstraction. This approach enables components exchange and component-based experiment. Additionally, a template model is provided for components which are frequently used in the manipulator model. In such cases, the model has adjustable parameter sets to be configured based on its implementation. Finally, the manipulator model is developed in different abstraction layers (Figure 4). The lower layer provides a more detailed information in each component and the upper layer provides the general overview of the system.

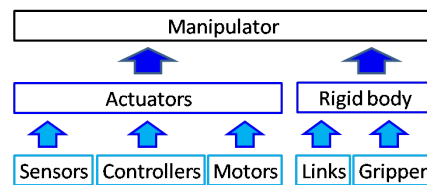


Figure 4: Abstraction layer in a manipulator model

In every modeling process, using estimated values and approximation is unavoidable mainly due to the following reasons:

- *Limited knowledge.* Many parameter set of a dynamic system are estimated through system identification (friction, inertia tensor).
- *Restricted information.* Many manufactures do not provide complete information about their product.

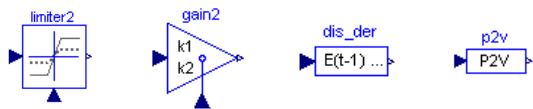
The use of estimated values and approximation is presented in the description of each package. The youBot Modelica library consists of four packages which are:

- *Controller* package,
- *Axis* package,
- *Body* package and
- *System* package.

The library is developed with the use of several packages in MSL such as Modelica.Blocks.Math for standard mathematical functions and Modelica.Mechanics for 3-dimensional mechanical systems. This paper follows the Modelica convention in describing the models. Model's name or package's name begins with capital letter. When necessary, the model includes its package name. The model *Modelica.Blocks.Interfaces.RealInput* refers to the model *RealInput* which is inside the package *Interfaces*. The *Interfaces* package is inside the *Blocks* package and the *Blocks* package is inside the MSL. An instance of a model is written in lower case aside from a few exceptional cases (e.g. *V* is used for voltage to differentiate from *v* for velocity).

### 4.1 Controller Package

The *Controller* package consists of the components for the manipulator control system. The *Controller* package is divided into three packages which are the *Components* package, the *PIDs* package and the *Modes* package. The *Controller.Components* package consists of models which are in the lowest level of abstraction layer. Figure 5 show the models in the *Controller.Component* package.



(a) *Limiter2* (b) *Gain2* (c) *DisDer* (d) *P2V*

Figure 5: The *Controller.Component* models

Following the Modelica convention, the instance's name of a model is placed on the upper part of the symbols in blue color. The model *Limiter2* and *Gain2* (Figure 5b and 5a) perform the calculation for non-linear PID controller (Equation 2). The model *DisDer* (Figure 5c) produces the derivative value of a specific time period from a discretized continuous input. The model *P2V* (Figure 5d) converts PWM rate to voltage rate. The *P2V* model is an approximation of the SVPWM component in the controller.

The *Controller.PIDs* package consists of three different PID models which are the *Position*, *Velocity* and the *Current* model. As the name suggests, the models are the PID controllers for position, velocity and current in the youBot manipulator (Figure 2). As a representative, Figure 6 shows the *PIDs.Velocity* model.

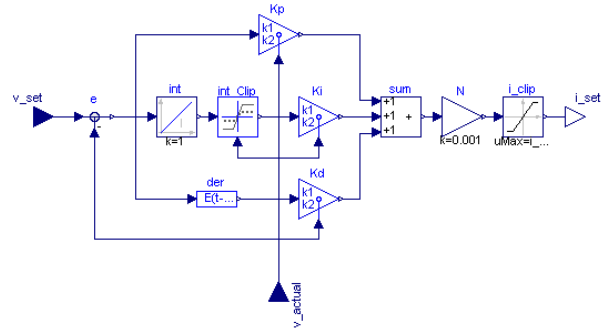


Figure 6: *PIDs.Velocity*

Where  $v_{set}$ ,  $v_{actual}$  and  $i_{set}$  represent  $v_{set}$ ,  $v_{actual}$  and  $i_{set}$  in Figure 3 respectively. The additional component  $N$  in the model produces the output in mA to mimic the readings of the actual system.

Finally, the *Controller.Modes* package is for different types of control mode. Currently, the available model in the *Modes* package is the *Position* model. Figure 7 shows the *Modes.Position* model.

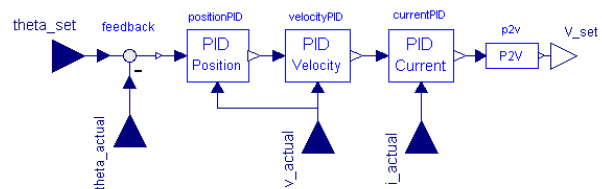


Figure 7: *Modes.Position*

The *Modes.Position* model consist of all three PID models from the *Controller.PIDs* package.  $V_{set}$  represent the voltage value which will be connected to the motor's power supply unit. Using the same approach, the model for other control mode explained in Section 3.2 can also be developed.

### 4.2 Axis Package

The *Axis* package consists of the model for joint actuator (motor and control system). The package is named *Axis* because the model will be connected to the rotating axis of the manipulator's joints. The *Axis* package consists of the *Position* model shown in Figure 8.



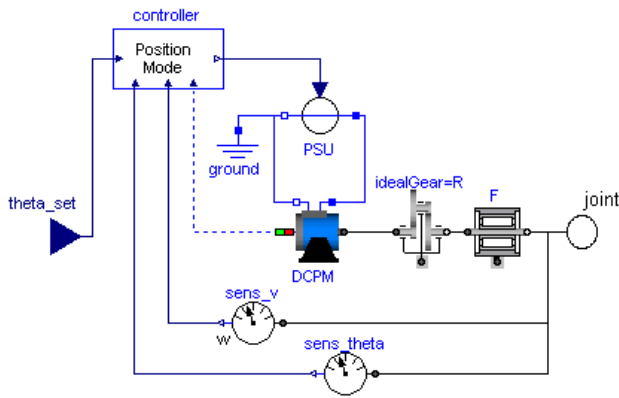


Figure 8: *Axis.Position*

Where *DCPM* represents the brushless DC motor model, *PSU* represents the power supply unit model, *R* represents the gearbox model, *F* represents the friction model, *controller* is the *Modes.Position* model (Figure 7), *sens\_v* represent the joint’s velocity sensor, *sens\_theta* represent the joint’s position sensor and *joint* is the connector to the manipulator’s joint model. The *controller* output (*V\_set* in Figure 7) is connected to *PSU* and its input is extended for the model input as *theta\_set*. The output of *sens\_v*, *sens\_theta* and the value of *DCPM*’s current is connected to the control system (*theta\_actual*, *v\_actual* and *i\_actual* in Figure 7). Aside from the controller, the component in *Modes.Position* are from MSL.

### 4.3 Body package

The *Body* package consists of models for the rigid body model of the manipulator’s kinematic chain. The *Body* package has three models which are *Gripper*, *Link* and *Manipulator*. The *Body.Gripper* model is the rigid body model of the youBot two finger gripper. Figure 9 shows the *Body.Gripper* model.

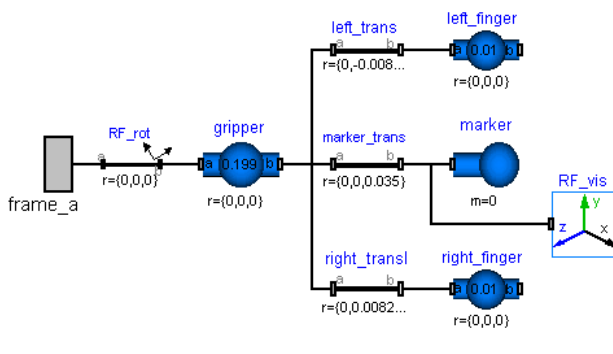


Figure 9: *Body.Gripper*

Where *RF\_rot* is the gripper’s reference frame rotation, *frame\_a* is the connector to the previous link model, and *gripper*, *left\_finger* and *right\_finger* are the rigid body model of the gripper body, left finger and right finger respectively. *marker* is a weightless body model to visualize the path of the manipulator’s end effector in simulation and *RF\_vis* provides the visualization of its reference frame.

The *Body.Link* model is the rigid body model of a manipulator link. Figure 10 shows the *Body.Link* model.

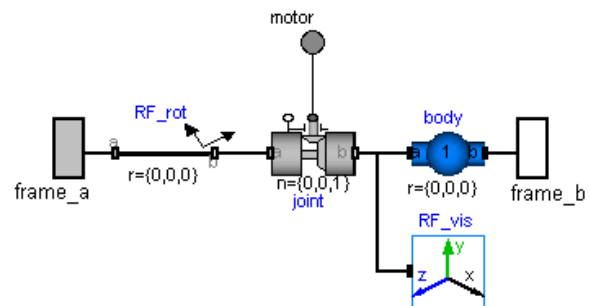


Figure 10: *Body.Link*

Where *joint* represent the link’s joint which is connected to the actuator model through the connector *motor*, *frame\_b* is the connector to the next link model and the *body* represent the rigid body of the link. *frame\_a*, *RF\_rot* and *RF\_vis* represent the same components as in *Body.Gripper* model.

The *Body.Manipulator* model represent the rigid body model of the youBot manipulator’s kinematic link. Figure 11 shows the *Body.Manipulator* model.

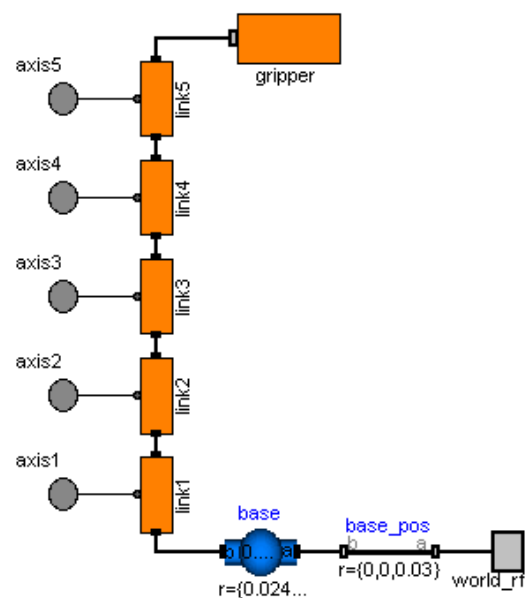
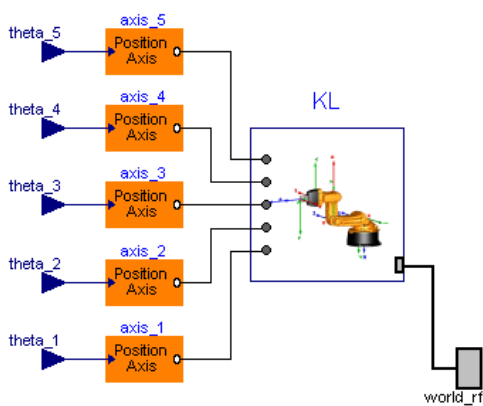


Figure 11: *Body.Manipulator*

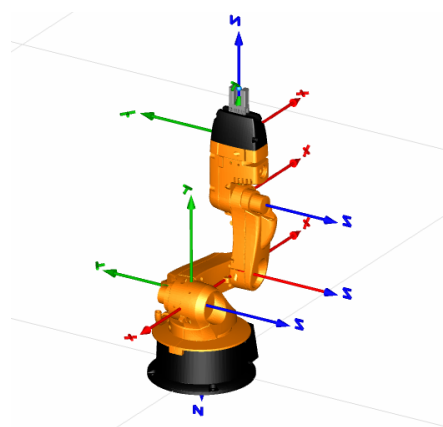
Where *link1* represent the first link of the manipulator (*Body.Link*, Figure 10), *gripper* is the manipulator’s gripper model (*Body.Gripper*, Figure 9), *base* represent the rigid body of the manipulator’s base. The component *base\_pos* is for defining the manipulator position in the world reference frame. The *Body.Manipulator* model has five connectors (*axis1* to *axis5*) for each joint model and one connector (*world\_rf*) for the world reference frame.

### 4.4 System package

The *System* package consists of the manipulator ready-to-use models. The *System* package has two models which are the *Position* model and the *Dummy* model. The *System.Position* model is the rigid body model of youBot manipulator’s kinematic chain and its actuators. Figure 12 shows the *System.Position* model and its visualization in Dymola whereas Figure 13 shows the parameter set configuration for the velocity controller in the manipulator’s fifth joint.



(a) *System.Position*



(b) Model visualization

Figure 12: The youBot manipulator model

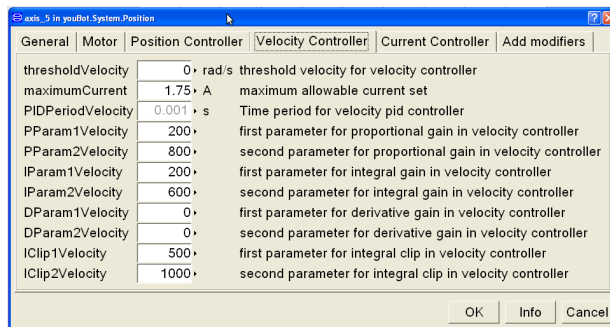


Figure 13: Parameters configuration

In Figure 12a, *KL* represents the rigid body model of the youBot manipulator’s kinematic chain, *theta\_5* represents the user defined joint angle and *axis\_5* represents the actuator (Figure 8) for joint 5. The parameter names in Figure 13 are consistent with the existing driver and firmware. The *System.Dummy* model is the rigid body model of the youBot manipulator (*Body.Manipulator*, Figure 11) connected to dummy actuators (*Modelica.Mechanics.Rotational.Speed*). The user can set the velocity of each joint directly in the *System.Dummy* model. The *System.Dummy* model is used for comparison test in Section 5.

## 5 Comparison Test

A comparison test with the actual system is performed after the development of the manipulator model. The test purpose is to evaluate the model accuracy and identify the major components which require further development. The test involves the comparison of the joint position and the joint velocity throughout a point-to-point motion. For the actual system, the joint velocity is recorded while performing the motion. The sensor measurement of the joint velocity is assumed to be accurate. Afterward, the recorded joint velocity is used as input for the *System.Dummy* model. In the same setting, the manipulator model (*System.Position*, Figure 12a) is also performing the same motion.

The manipulator’s joints in this test are set to be frictionless. The motion involves all joints moving 90°. Such motion was chosen so that the resulting error will be the accumulation of the estimated value and approximation in all joints. Figure 14a shows the end-effector paths during the motion (the gray-colored youBot manipulator is the starting pose of the motion.) whereas Figure 14b shows the error in joint position. As expected, the path generated by the model is smoother than that of the actual system as a result of the idealistic conditions in the simulation. The sum

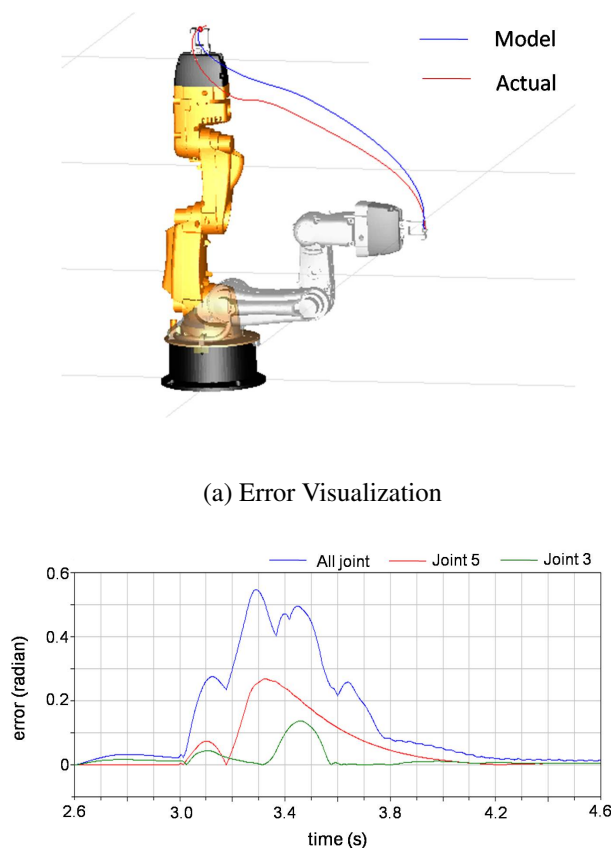
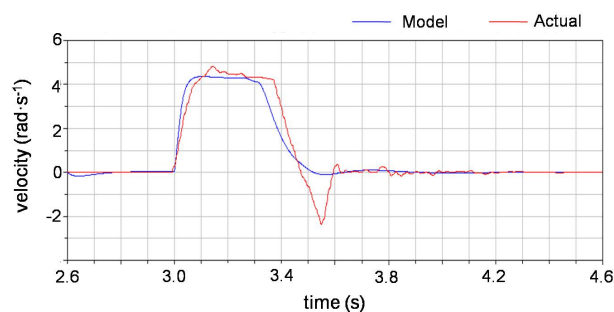


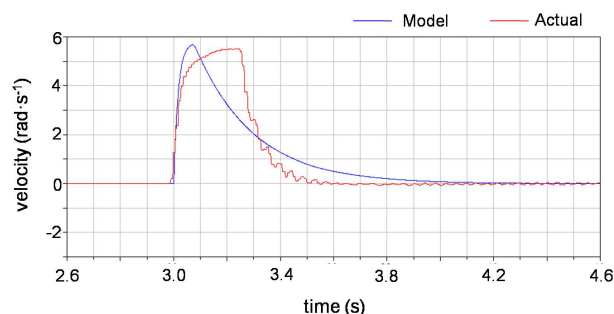
Figure 14: Test result

of error from all joint peaked at the value of 0.55 radian. The error in each joint depends on the maximum velocity parameter ( $v_{max}$ ) in the controller. As shown in Figure 14b, joint 3 ( $v_{max} = 4.19 \text{ rad} \cdot \text{s}^{-1}$ ) has a considerably lower peak than joint 5 ( $v_{max} = 5.90 \text{ rad} \cdot \text{s}^{-1}$ ). The error in joint angle peaked at two points. Both peak points happened slightly after the velocity change (from stop to move and slowing down from a constant velocity). This is consistent with the error in joint velocity as shown in Figure 15.

The joint velocity in the actual system is less stable than in the simulation (Figure 15a). This is the result of the motor vibration which is excluded from the manipulator model. The ideal motor model results in the deviation on higher velocity (Figure 15b) which correspond to the higher error in joint position for joints with higher  $v_{max}$  value in its controller. Similar phenomena in joint velocity and joint position are also found in other joints. Other possible contributing aspects in the deviation between the model and the actual system are the inertia tensor estimation, gearbox elasticity/damping, SVPWM approximation and frictionless joints.



(a) Joint 3



(b) Joint 5

Figure 15: Joint velocity comparison

## 6 Conclusion

In this paper, the development of Modelica model for the youBot manipulator is presented. The Modelica library for the manipulator components provides the user with modularity, reusability and abstraction. The model accuracy has been evaluated through a comparison test with the actual system. The test result shows that the model reflects the actual system within a reasonable deviation. Possible improvements for the developed Modelica library is the development of a more accurate motor model and a more comprehensive evaluation of the manipulator component (controller components, power consumption and dynamic properties of every rigid body model). The manipulator model is planned to be tested with other Modelica tools (OpenModelica, jModelica) and used for hardware-in-the-loop experiments. The development or design of other manipulator models is also possible through the reusability of the components model in the Modelica library. The library is publicly available<sup>1</sup> to be used for education or research involving manipulator dynamics, load identification, fault analysis and motion control.

<sup>1</sup>[www.youbot-store.com](http://www.youbot-store.com)

## References

- [1] R. Bischoff, U. Huggenberger, and E. Prassler, “Kuka youbot - a mobile manipulator for research and education,” in *IEEE Int. Conf. on Robotics and Automat. (ICRA)*, pp. 1–4, May 2011.
- [2] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, “Virtual robot experimentation platform v-rep: a versatile 3d robot simulator,” in *Proc. of the Second Int. Conf. on Simul., Model., and Program. for Auton. Robots, SIMPAR’10*, (Berlin, Heidelberg), pp. 51–62, Springer-Verlag, 2010.
- [3] O. Michel, “Webots: Professional mobile robot simulation,” *J. of Adv. Robotics Syst.*, vol. 1, no. 1, pp. 39–42, 2004.
- [4] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ Int. Conf on Intell Robots and Syst.*, pp. 2149–2154, 2004.
- [5] H. Elmqvist, S. Mattsson, and M. Otter, “Modelica-a language for physical system modeling, visualization and interaction,” in *Comput. Aided Control Syst. Des.. in Proc. of the 1999 IEEE Int. Symp. on*, pp. 630–639, 1999.
- [6] P. Fritzson, “Principles of object-oriented modeling and simulation with modelica 2.1,” pp. 19–67, 2004.
- [7] G. Ferretti, M. Gritti, G. Magnani, and P. Rocco, “A remote user interface to modelica robot models,” in *Proc. of the 3rd Int. Modelica Conf.*, pp. 231–240, 2003.
- [8] M. Thuemmel, M. Otter, and J. Bals, “Control of robots with elastic joints based on automatic generation of inverse dynamics models,” in *Proc. of 2001 IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 925–930, 2001.
- [9] A. Kazi, Merk, M. G. Otter, and H. Fan, “Design optimisation of industrial robots using the modelica multi-physics modeling language,” in *Proc. 33rd ISR Int. Symp. Robot*, pp. 347–352, 2002.
- [10] M. Hast, J. Åkesson, and A. Robertsson, “Optimal robot control using modelica and optimica,” in *Proc. of the 7th Int. Modelica Conf.*, Modelica Association, Sept. 2009.
- [11] I. Dressler, J. Schiffer, and A. Robertsson, “Modeling and control of a parallel robot using modelica,” in *Proc. 7th Int. Modelica Conf.*, (Como, Italy), Sep 2009.
- [12] Y. Chen, S. Jin, X. Zou, D. Xu, and W. Cai, “Study of modeling and simulating for picking manipulator based on modelica,” in *Proc. of the 2nd Int. Conf. on Intell. Robotics and Appl.*, (Berlin, Heidelberg), pp. 1211–1216, Springer-Verlag, 2009.
- [13] J. Åkesson, U. Nordstroem, and H. Elmqvist, “Dymola and modelica\_embeddedsystems in teaching - experiences from a project course,” in *Proc. of the 7th Modelica Conf.*, 2009.
- [14] U. Pohlmann, S. Dziwok, J. Suck, B. Wolf, C. Loh, and M. Tichy, “A modelica library for real-time coordination modeling,” in *Proc. of the 9th Int. Modelica Conf.*, 2012.

## A Manipulator Specification

Table 1: Kinematic chain

	Parent frame	Translation (cm)			Rotation (degree)		
		x	y	z	x	y	z
Joint 1	Base	2.4	0	11.5	180°	0°	0°
Joint 2	Joint 1	3.3	0	0	90°	0°	-90°
Joint 3	Joint 2	15.5	0	0	0°	0°	-90°
Joint 4	Joint 3	0	13.5	0	0°	0°	0°
Joint 5	Joint 4	0	11.36	0	-90°	0°	0°
Gripper	Joint 5	0	0	5.716	90°	0°	180°

Table 2: Joint range

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5
Joint range	-169°	-65°	-151°	-102.5°	-165°
	169°	90°	146°	102.5°	165°

Table 3: Dynamic Properties

	Mass (kg)	Inertia Tensor Elements ( $kg \cdot cm^2$ )		
		$I_{xx}$	$I_{yy}$	$I_{zz}$
Link 1	1.39	29.525	60.091	58.821
Link 2	1.318	31.145	5.483	31.631
Link 3	0.821	17.2767	4.1967	18.468
Link 4	0.769	6.764	10.573	6.61
Link 5	0.678	1.934	1.602	0.689
Gripper	0.201	2.324	3.629	2.067

# Equation based parallelization of Modelica models

Marcus Walther Volker Waurich Christian Schubert Dr.-Ing. Ines Gubsch  
Dresden University of Technology  
{marcus.walther, volker.waurich, christian.schubert, ines.gubsch}@tu-dresden.de

## Abstract

In order to enhance the performance of modern computers, the current development is towards placing multiple cores on one chip instead of increasing the clock rates. To gain a speed-up from this architecture, software programs have to be partitioned into several independent parts. A common representation of these parts is called a task graph or data dependency graph. The authors of this article have developed a module for the OpenModelica Compiler (OMC), which creates, simplifies and schedules such task graphs. The tasks are created based on the BLT (block lower triangular)-structure, which is derived from the right hand side of the model equations. A noticeable speed-up for fluid models on modern six-core CPUs can be achieved.

*Keywords: modelica; openmodelica; parallelization; BLT, task graph*

## 1 Introduction

Modelica has become a widely used standard to describe physical simulation models. Compiling such a model into binary code can be performed by applications like Dymola, SimulationX or OpenModelica. However, all these tools only create a single thread simulation code out of standardized Modelica models, which does not allow for a speed-up with modern multi-core CPUs. This is due to the dependencies among the model equations which have to be considered in order to distribute the tasks amongst several threads.

The approaches to parallelize Modelica models can be divided into manual and automatic parallelization. Manual approaches comprise the *parModelica* language extension [1] or the TLM technique [2]. In this paper, manual parallelization shall not be pursued further as it is not suitable for parallelizing existing models. A lot of effort has been spent on automatic parallelization methods. Peter Aronsson [3] presented a method based on fine grained task graphs

which were derived from the expressions of the model equations. Later, this approach was adapted by other authors (see for example [4], [2] and [5]), to perform simulations on Cell- and GPU-Architectures. Handling fine grained task graphs is a complicated and time consuming topic. Therefore additional work was required to reduce the graph complexity, for example with the help of a graph rewriting system [3]. This paper follows the ideas of Casella [6] who suggested to build a task graph parallelization based on the BLT representation of a model. He then also showed that, in case of fluid applications, this approach will lead to task graphs which can be well parallelized. Therefore, this paper explores the implementation of the ideas of [6] into the OpenModelica compiler. To evaluate the efficiency of the implementation, the idea of the *Maximum Theoretical Speedup* is introduced. Afterwards, different scheduling algorithms are presented which are required to assign each task to a thread. It is followed by a number of benchmarks which compare the effectiveness of the different scheduling algorithms and reveal further properties of different domains with respect to parallelization.

## 2 Parallelization of model equations

The equations of a simulation model are typically described as a set of Differential Algebraic Equations (DAEs). Equation 1 shows the basic definition of such a DAE.

$$F(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{v}, t) = 0 \quad (1)$$

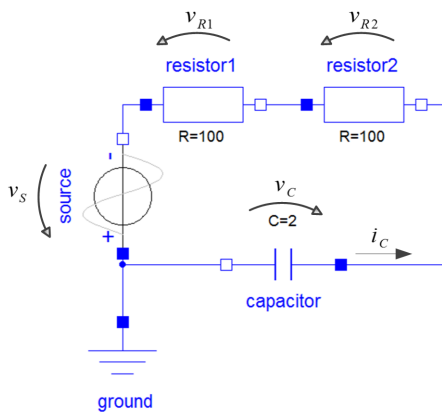
The value  $t$  represents the time. The vector  $\mathbf{x}$  holds all variables of the system whose derivatives with respect to time appear inside the equations. The derivatives itself are stored in  $\dot{\mathbf{x}}$ . In addition,  $\mathbf{v}$  contains all other algebraic variables. By applying index reduction, (1) is converted into a DAE with index one or zero. The Underlying Ordinary Differential Equation (UODE) contains all equations and variables necessary to calculate the reduced state set  $\dot{\mathbf{y}} \subseteq \dot{\mathbf{x}}$  of the model (see equation

2) and other equations (see equation 3) [6].

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \tag{2}$$

$$\mathbf{v} = \mathbf{g}(\mathbf{y}, t) \tag{3}$$

The equations and variables of the ODE system can be organized as an incidence matrix, with each matrix-row representing one equation and each column one variable [7]. If a variable is part of an equation, the matrix entry is filled with a value. A simple electric circuit, containing a power supply and two resistors as well as a capacitor, is displayed in figure 1. It can be described by the equations below. The rows and



$f_1 : v_s =$	<i>offset</i>
$f_2 : v_{R1} =$	$R_1 \cdot i_c$
$f_3 : P_{R1} =$	$v_{R1} \cdot i_c$
$f_4 : v_{R1} =$	$v_s - v_{R1n}$
$f_5 : i_c =$	$C \cdot \dot{v}_c$
$f_6 : v_{R2} =$	$R_2 \cdot i_c$
$f_7 : P_{R2} =$	$v_{R2} \cdot i_c$
$f_8 : v_{R2} =$	$v_{R1n} - v_c$

Figure 1: Simple model of an electrical circuit

columns of the incidence matrix can be arranged in a way that the matrix forms a block lower triangular matrix (BLT). Thus, the blocks of equations can be solved from the top to the bottom via forward substitution. First, the power supply voltage  $v_s$  is calculated from equation  $f_1$ . After that, the equations  $f_4, f_8, f_6$  and  $f_2$  of the circuit cannot be calculated as single equations, as they have two unknown variables. More precisely they are forming a circular dependency, because the variable  $u_{R1}$  is solved in equation  $f_2$ , which requires variable  $i_c$ , solved by equation  $f_6$ , for calculation. Furthermore, the calculation of  $f_6$  depends on the equation  $f_8$  which depends on equation  $f_4$ . And

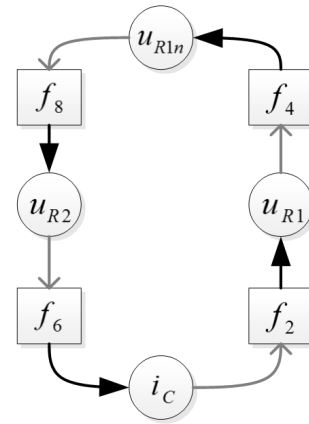


Figure 2: A bipartite graph representing the circular dependency between the equations  $f_2, f_4, f_6$  and  $f_8$

finally the equation  $f_4$ , solving variable  $u_{R1n}$ , requires  $u_{R1}$ , still solved by  $f_2$ . This fact is shown in Figure 2. That is why they have to be handled in an equation system which combines all equations into one block (see figure 3, gray coloured box). As shown in the example, blocks can be very simple, containing just one single equation or they can be really complex, containing hundreds of equation stored in an equation system. In order to solve the system efficiently, the block size

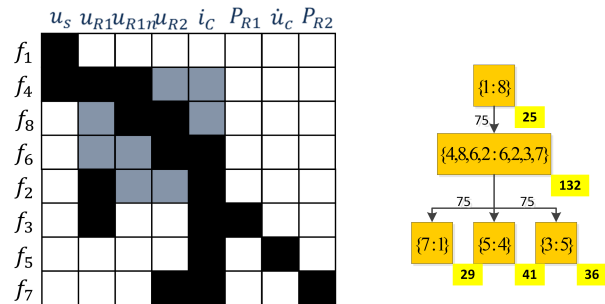


Figure 3: BLT-Matrix of the example on the left, derived task graph on the right side

should be as small as possible. To find the smallest blocks of the system, Tarjan's algorithm [8] can be used. A complete algorithm to get the blocks from the DAE-System is presented in [6].

### 3 Task Graph representation

A task graph or data dependency graph is a widely used technique to describe the different parts of a program and their relationships among each other. The graph contains nodes representing the tasks and directed edges. If an edge goes from node  $n_1$  to  $n_2$ , the task  $n_1$  has to be executed before the task  $n_2$  can start.

Parallel branches of such a graph can be calculated in parallel, because there are no direct dependencies between them and thus they could be handled by different threads.

To get a task graph out of the BLT-structure, all blocks of the matrix are converted into a node of the graph. After that, the calculation dependencies between the blocks have to be inserted. An edge between the nodes of the blocks  $n_i$  and  $n_j$  is added to the graph, if the BLT-matrix has an entry at position  $(i, j)$ , with  $i > j$  and  $(i, j)$  representing the row and column index of the blockmatrix, respectively.

The task graph of the given example circuit is displayed in figure 3. The notation inside the nodes is  $\{equation\ index : variable\ index\}$ . For the given example, the last three tasks could be handled in parallel by three different threads. In order to evaluate the simulation speed, both execution and communication costs for the tasks and processors have to be known. The execution cost of a task is the number of cycles or the time span required to calculate it. Communication costs are the time to transfer all required variables from one thread to another. To measure these values, two benchmark programs were developed. To estimate the communication costs, a standalone benchmark has been created which copies different sized data arrays of 64 bit long floating point value from one thread to another. By analyzing the task equations, the number of variables, which are transferred by each edge in the task graph, can be obtained. Thus a communication cost estimate can be assigned to each edge. The execution costs are being measured with the help of the OpenModelica *measure time* functionality for a serial calculation run, which creates a xml-file containing execution times for each block. This approach has still some drawbacks. First, it cannot be exact, as it is not possible to predict the occurrence of cache misses or context switches between different processes. Second, one serial execution of the model is required, which may cause a severe overhead, before the estimates are available.

Execution costs are displayed on the bottom right corner of the nodes on a yellow background while communication costs are displayed near the edges, see Figure 3.

## 4 Graph simplification

Complex Modelica models may lead to complex task graphs with thousands of nodes and edges. Scheduling these graphs (see section 5) is a time consuming

process, which can, depending on the scheduling algorithm, scale superlinear with the number of nodes and edges. Therefore the authors have implemented two rules to simplify complex task graphs, based on the ideas of [3]. The first one is a simple rule to merge chains of nodes with a maximum of one successor and one predecessor into one, as there is no point in calculating these nodes by different threads. The rule is called "mergeSimpleNodes" and an example is displayed in figure 4. The second rule, which is more

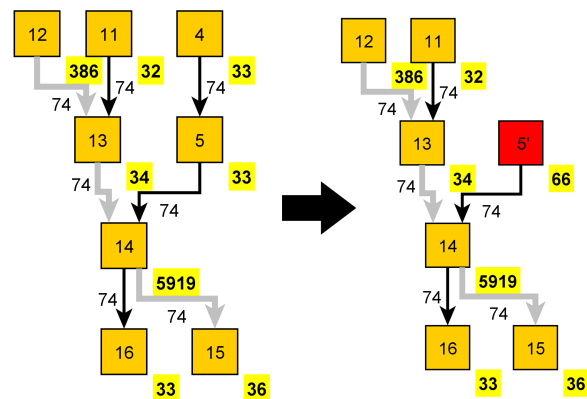


Figure 4: Example of the "mergeSimpleNodes" rule. Node four and five are merged into one.

complex, is called "mergeParentNodes". It consolidates a node with its parents, if this leads to an decreasing execution time. See figure 5 for an example. If the nodes 11 and 13 are handled by different threads than task 12, the execution time increases compared to the serial execution. To prevent this, the nodes are merged into task 13'.

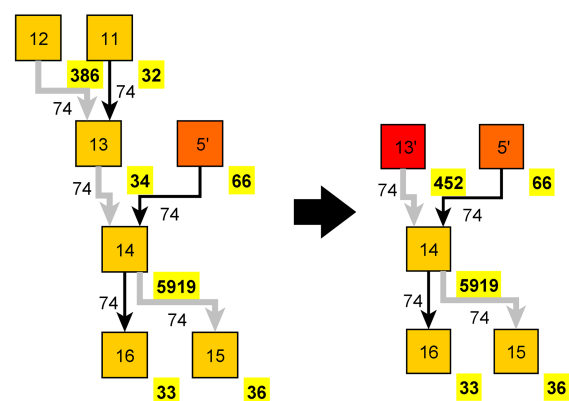


Figure 5: Example of the "mergeParentNodes" rule. Node 11, 12 and 13 are merged into one.

## 5 Task Graph Scheduling

Based on the derived task graph, the different tasks can be dispersed among different threads, which can later be distributed on different processors or processor-cores. This is called Scheduling. In the presented work the Scheduling is performed during compile time (static scheduling). If the mapping between tasks and threads is set during run time, it is called a dynamic scheduling.

In order to achieve a proper mapping, most of the static scheduling algorithms use the information about the execution and communication costs to load the threads evenly and with shortest idle time. Finding the ideal schedule is a NP-hard problem [9]. Thus, different heuristics are used which have been implemented into the OpenModelica compiler module. To analyse the scheduling of a task graph several evaluation parameters can be obtained. First, some basic definitions shall be given. The *serial time*  $t_S$  of a model is the sum of all execution costs of all tasks  $T$

$$t_S = \sum_{i \in T} t_i \quad (4)$$

The *minimum parallel time*  $t_{Pmin}$  is equal to the sum of the execution costs along the critical path, denoted as *crit'*

$$t_{Pmin} = \sum_{j \in \text{crit}'} t_j \quad (5)$$

This definition neglects all communication costs and corresponds to the case where all tasks of the critical path are assigned to the same thread. Further, it is assumed that all other tasks are handled in parallel by other threads not causing any delays. Clearly, this would require a sufficiently large number of computing cores. The *parallel time*  $t_P$  accounts for a limited number of computing cores and denotes the time required to calculate all tasks of a task graph given a schedule (assignment for each task to a thread or computing core) considering both execution as well as communication costs. The *Maximum Theoretical Speed-up*  $n_{max}$  can be obtained by dividing the serial time by the minimum parallel time

$$n_{max} = \frac{t_S}{t_{Pmin}} \quad (6)$$

assuming that an infinite number of computing cores is available. The *Theoretical Speed-up*  $n_t$  provides the expected speed-up for a given schedule. It is defined as the serial time divided by the parallel time

$$n_t = \frac{t_S}{t_P} \quad (7)$$

This quantity shall be used to compare different scheduling algorithms and to evaluate the implementation of the parallel code. In the following, different scheduling algorithms which will be compared are presented.

### 5.1 Level Scheduling

The simplest implemented scheduling algorithm is the level scheduling, which divides the graph into several layers. All tasks of one layer have just dependencies to tasks of previous layers. Thus, no direct dependencies between tasks of the same layer are allowed. The tasks of each layer are calculated in parallel until the algorithm proceeds with the next layer. Figure 6 shows a small graph example. Each layer is implemented as one OpenMP-Sections-Region and each task is handled in one OpenMP-Section. An example is displayed in listing 1.

Listing 1: Level scheduling code for graph in figure 6

```
static void solveODE(data) {
  //Level 1
  #pragma omp parallel sections {
    #pragma omp section {
      eqFunction_12(data);
    }
    #pragma omp section {
      eqFunction_11(data);
    }
    #pragma omp section {
      eqFunction_4(data);
    }
  }
  //Level 2
  #pragma omp parallel sections {
    #pragma omp section {
      eqFunction_13(data);
    }
    #pragam omp section {
      eqFunction_5(data);
    }
  }
  //Level 3
  #pragma omp parallel sections {
    #pragma omp section {
      eqFunction_14(data);
    }
  }
}
```

Therefore, OpenMP takes care of the concrete mapping between tasks and processors. This approach is following the ideas of a breadth-first-scheduling [10]. An eminent advantage of this scheduling method is that no information about execution and communication costs is needed. The scheduling is only based on the task dependencies. Thus, this algorithm is not fully static, but a hybrid of static and dynamic scheduling.

### 5.2 List Scheduling

The authors have also implemented a simple list scheduling algorithm [11], which can handle the tasks from root-nodes to leaf-nodes or vice versa. The list scheduling algorithm performs in the following way.



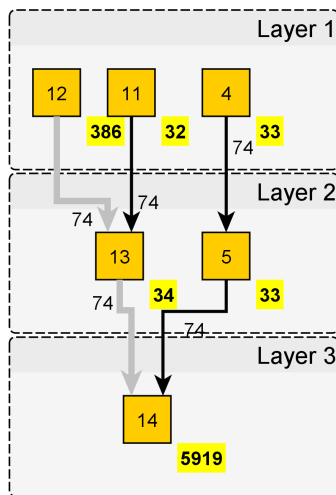


Figure 6: Example of level scheduling.

The root-nodes of the task graph are collected in a so-called 'ready list'. These tasks are distributed to all available threads. Before the first task assignment, every thread has a ready time of zero which means it is considered to be idle. If a task is assigned to an idle thread, the execution costs of the task will be added to its ready-time. The assignment of tasks enables the scheduling of their successor nodes which will be appended to the ready-list. The tasks from the ready list will be distributed successively to the thread with the earliest ready time. If the predecessor of an assignable task is scheduled to the same thread, the communication costs between these tasks are not taken into account. Otherwise, they have to be added to the ready time of the thread. The algorithm terminates when the ready list is empty.

### 5.3 Modified Critical Path Scheduling

Static scheduling algorithms are reviewed thoroughly. There is a multiplicity of approved scheduling heuristics and each performance depends on the structure of the graph, the dispersal of the costs etc. As an exemplary method, the 'Modified Critical-Path Scheduler' (MCP) by Wu and Gajski [12] has been implemented since this one is well-established as a reference heuristic[13]. The MCP distributes the tasks successively like the list scheduling to the thread that allows its earliest execution. The prioritisation of the assignable tasks is based on their ALAP-binding. The ALAP-binding stands for the as-late-as-possible start time and is computed as the longest path from the task to the finishing time of the last executed task.

### 5.4 External Scheduling

In order to understand and test the effect of different schedulings, the authors implemented a manual graph scheduler. The tasks can be assigned to the threads by hand on a graphical interface. For instance, the graph can be divided into vertical stripes, each handled by one thread like performed by libraries like metis [14].

### 5.5 Deadlock Detection

To check if a schedule is free of deadlocks, a transformation into a state / transition - petri net was developed. Every task of the graph is transformed into two states and one transition and every edge to one transition. An example for such a transformation is given in figure 7, where the task graph is shown on top and the petri net on bottom. The tasks of each thread are displayed one below the other. This kind of view gives more detailed information about the required locks between the different threads. If the final states of all threads are connected to a transition that is connected to the first states (not displayed in the figure), a petri net tool can check if the graph is free of deadlocks.

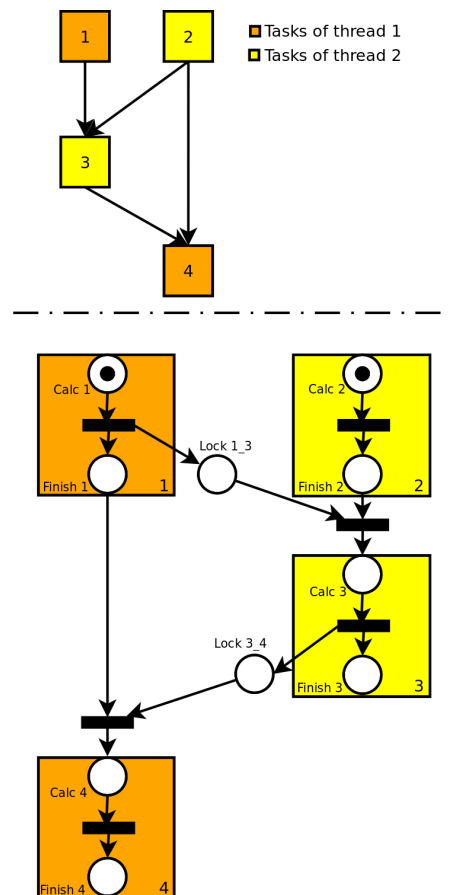


Figure 7: Example of state / transition net transformation

## 6 Benchmarks

To compare the different scheduling algorithms with the serial code, the simulation time of various models was measured. Three representative models from the domains Mechanics, Fluid and Electrics were selected out of the modelica standard library "MSL32". The first one is the engine V6 mechanic model. The second is the branching dynamic pipes example of the fluid domain. And the last model is the electrical cauer low pass sc model.

The test system was a computer with an Intel Core i7-3930K with six cores @ 3.20GHz and 32 gigabyte RAM running Windows 7 professional. All models were simulated from 0s to 1s using the dassl-solver.

In order to evaluate if it is possible to achieve shorter simulation times, the theoretical maximum speed-ups are displayed in table 1 for the three models.

The generated code of all scheduling algorithms, except the level-scheduling, is realized with pThreads using spin locks. Level-scheduling is based on an OpenMP implementation, as described in the previous section. Unfortunately, the results of level-scheduling were considerably slower than the other algorithms. Hence they were omitted from the diagrams for the sake of clarity.

Table 1: *Maximum Theoretical Speed-up  $n_{max}$*

Modell	$n_{max}$
Engine V6	1.11
BranchingDynamicPipes	13.09
CauerLowPass	5.97

For the mechanical model it is not possible to achieve a speed-up at the moment. This is due to the graph's structure. Every multibody system requires the solution of the following linear system [15]

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{f}^a + \mathbf{G}^T(\mathbf{q})\mathbf{f}^c \quad (8)$$

$$\mathbf{0} = \mathbf{g}(\mathbf{q}) \quad (9)$$

The mass matrix  $\mathbf{M}$  is in general densely populated. Its number of rows and columns is equal to the degrees of freedom of all the *tree-joints*, see [15]. The authors have noted that a large portion of the calculation time is spent on solving this linear system which corresponds to a single node in the task graph. In case of the EngineV6 model this can be up to 95% of the entire execution time. To take advantage of the BLT approach for such a model, one would have to either find a way to split up this task into smaller ones or to solve the task itself in parallel. This will be a topic of

further research. For the fluid and electrical models, a speed-up is theoretically possible.

The benchmark results are displayed in the figures 8, 9 and 10. As already assumed, no speed-up with the mechanic model was found. The results of the fluid

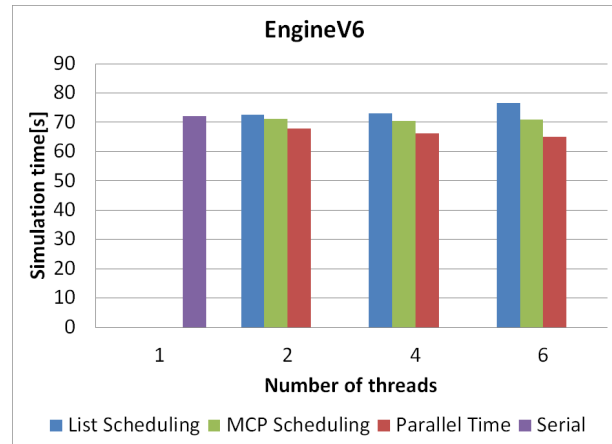


Figure 8: Benchmark of the engineV6 example

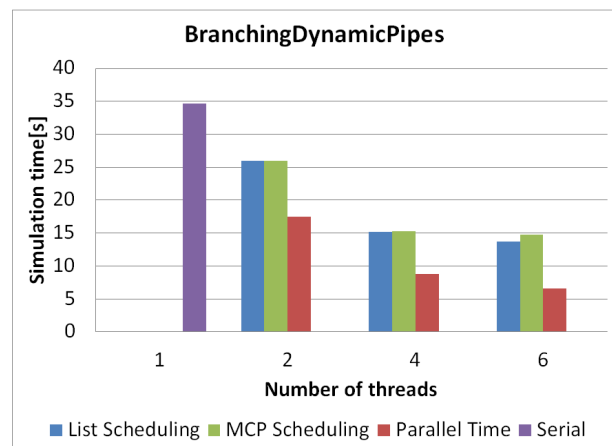


Figure 9: Benchmark of the dynamic pipes example

benchmarks indicates significant enhancement. The simulation showed a large step size, resulting in a lower number of calculations of the ODE functions. Carrying out such an ODE calculation takes a long time compared to the other examples. Moreover, the task graph has a lot of tasks which can be calculated in parallel. The trend of the Parallel Time indicates further potential. Additional research is needed in order to close the gap between predicted and measured speed-up. The low pass example shows currently no significant speed-up for the BLT parallelization, although the task graph has a lot of tasks in parallel, as can be seen in the behaviour of the Parallel Time. The issue appears to be the short execution time of the entire ODE system. The calculation of this sys-

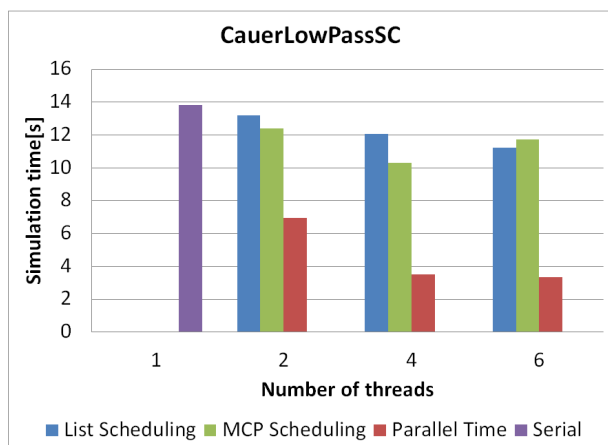


Figure 10: Benchmark of the cauer low pass example

tem is about 30 times faster than the calculation of the branching dynamic pipes ODE system. The overhead of the parallel code seems to be too big to cope with such short calculation cycles. At the beginning of the presented work, the parallel code was solely implemented with OpenMP. In consequence of the bad measurements of the cauer low pass example, the OpenMP code was exchanged with pThreads code using spin locks. The execution time of the parallel code could be halved for the cauer low pass, but is still too large to achieve a significant speed-up for the model.

## 7 Conclusion

The implementation has shown that the BLT parallelization approach is able to reduce the simulation time for some models, especially if they are part of the fluid domain. To achieve speed-ups for various models, further research is required, especially to reduce the overhead of the parallel code and to handle one big task using multiple cores. The problem with the big tasks can be solved by applying parallel solvers or by splitting up the complex task into simpler ones.

Furthermore, some memory analysis needs to be performed to reduce the number of cache misses and invalidations between the threads. At the moment, the variables are stored regarding their types (real, int or boolean) in different arrays. To get some improvements, the variables have to be organized regarding their task affinity.

All in all the presented BLT approach looks promising for a parallelization speed-up on ordinary shared memory systems.

## References

- [1] M. Gebremedhin, “Parmodelica: Extending the algorithmic subset of modelica with explicit parallel language constructs for multi-core simulation,” *Linköping University Electronic Press, Linköpings universitet*, 2011.
- [2] K. Stavåker, “Contributions to parallel simulation of equation-based models on graphics processing units,” *Linköping University, Sweden*, 2011.
- [3] P. Aronsson, *Automatic Parallelization of Equation-Based Simulation Programs*. Institutionen för datavetenskap, 2006.
- [4] H. Lundvall, K. Stavåker, P. Fritzson, and C. Kessler, “Automatic parallelization of simulation code for equation-based models with software pipelining and measurements on three platforms,” *SIGARCH Comput. Archit. News*, vol. 36, pp. 46–55, June 2009.
- [5] P. Östlund, “Simulation of modelica models on the cuda parallel architecture,” *Linköping University Electronic Press, Linköpings universitet*, 2010.
- [6] F. Casella, “A strategy for parallel simulation of declarative object-oriented models of generalized physical networks,” *Linköping University Electronic Press, Linköpings universitet*, 2013.
- [7] F. Cellier and E. Kofman, *Continuous System Simulation*. Springer, 2006.
- [8] R. E. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.
- [9] M. R. Garey, D. S. Johnson, and R. Sethi, “The complexity of flowshop and jobshop scheduling,” *Math. of Op. Res.*, vol. 2, pp. 117–129, May 1976. ctr127.
- [10] A. Duran, J. Corbalán, and E. Ayguadé, “Evaluation of openmp task scheduling strategies,” in *IWOMP* (R. Eigenmann and B. R. de Supinski, eds.), vol. 5004 of *Lecture Notes in Computer Science*, pp. 100–110, Springer, 2008.
- [11] U. Banerjee, “Parallelization, basic block,” in *Encyclopedia of Parallel Computing* (D. A. Padua, ed.), pp. 1450–1458, Springer, 2011.

- [12] M.-Y. Wu and D. D. Gajski, “Hypertool: A programming aid for message-passing systems,” *IEEE TRANS. ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol. 1, pp. 330–343, 1990.
- [13] A. Radulescu and A. J. C. van Gemund, “Flb: Fast load balancing for distributed-memory machines.,” in *ICPP*, pp. 534–541, 1999.
- [14] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, Dec. 1998.
- [15] H. Elmqvist and M. Otter, “Methods for tearing systems of equations in object-oriented modeling,” in *ESM*, vol. 94, p. 1–3, 1994.

# Simulation of 2-dimensional flows in Modelica with the Cascaded Digital Lattice Boltzmann Method

Thomas Bäuml Helmut Kühnelt  
 AIT Austrian Institute of Technology GmbH  
 Mobility Department, Electric Drive Technologies  
 Giefinggasse 2, 1210 Vienna, Austria

*Keywords: Modelica, Cascaded Digital Lattice Boltzmann, 2-dimensional flows*

## Abstract

This paper deals with the implementation of a general methodology for modeling two-dimensional fluid flows in Modelica applying the Cascaded Digital Lattice Boltzmann Method. This approach models fluid flow as collective dynamics of fictitious particles on the nodes of a regular lattice. The various elements needed for simulation are described in Modelica and generic test cases are set up. The method is able to deal with simple scenarios where the powerful capabilities of advanced CFD tools are not needed.

## 1 Introduction

Calculating the dynamics of fluid flows is an important topic in the field of simulation. Common practice is to simulate complex scenarios by utilizing Computational Fluid Dynamics (CFD). Despite its capability of representing fluid flows in a very detailed way it has the drawback of compatibility. Coupling with other physical domain simulations is only possible by co-simulation. In this contribution a general methodology for modeling two-dimensional fluid flows in Modelica is shown. Whereas in [1] the Navier-Stokes equations are solved by a finite volume method, this work deals with modeling them with a Lattice Boltzmann Method (LBM).

## 2 Theory

The Lattice Boltzmann method is a relatively new simulation technique for fluid systems that has attracted interest as alternative to the discretization of

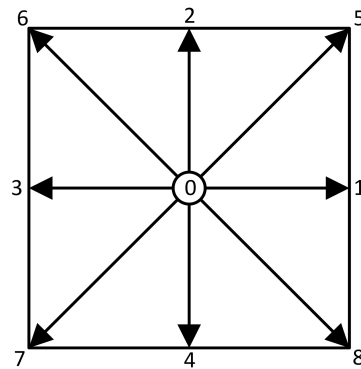


Figure 1: Lattice Boltzmann D2Q9 grid element

the Navier-Stokes equations. Instead of discretizing the Navier-Stokes equations to solve the conservation equations of macroscopic quantities (i.e., mass, momentum, and energy), LBM is a mesoscopic approach for modeling macroscopic fluid dynamics based on the Boltzmann kinetic equation which describes the statistical behavior of a non-equilibrium thermodynamic system. In the LBM, the fluid motion is based on the collective dynamics of fictitious particles on the nodes of a regular lattice. The dynamics of these particles is designed to obey the basic conservation laws ensuring hydrodynamic behavior in the continuum limit. The basic quantity is the particle distribution function  $f_i(\vec{x}, t)$  that represents the probability of finding a fluid particle density  $i$  at a location  $\vec{x}$  and at a time  $t$  traveling with a discrete speed  $\vec{c}_i$ .

The mass density  $\rho$  and the momentum density  $\rho\vec{v}$  are given by:

$$\rho(\vec{x}, t) = \sum_{i=0}^n f_i(\vec{x}, t) \quad (1)$$

$$\rho(\vec{x}, t) \vec{v}(\vec{x}, t) = \sum_{i=0}^n f_i(\vec{x}, t) \vec{c}_i \quad (2)$$

The motion of the particles is restricted to the node positions of a regular lattice. In 2D, commonly a 9-speed, quadratic lattice (D2Q9) with mesh spacing  $\Delta x$  is applied, where the discrete velocities  $\vec{c}_i$  connect lattice nodes to first and second neighbors and which has a rest particle  $f_0$ , see Figure 1. Here,  $c_x = [0, 1, 0, -1, 0, 1, -1, -1, 1]^T$  and  $c_y = [0, 0, 1, 0, -1, 1, 1, -1, -1]^T$ .

The spatial and temporal evolution of the particle distribution function is described by an explicit discretization of the Boltzmann equation, given by the following equation:

$$f_i(\vec{x} + \vec{c}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t) - \omega (f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)) \quad (3)$$

There, the left-hand side represents the molecular free streaming from one lattice node to the other, whereas the right-hand side represents the molecular collisions via a single-time relaxation towards local equilibrium  $f_i^{eq}$  on a typical timescale  $\tau = 1/\omega$ .  $\tau$  is related to the macroscopic kinematic viscosity  $\nu = c_s^2 \Delta t (\tau - 1/2)$ , where  $c_s = 1/\sqrt{3}$  is the speed of sound. Commonly  $\Delta t^{(LB)} = 1$  in lattice units, thus rendering the grid spacing  $\Delta x^{(LB)} = 1$ . Results in physical units can be obtained by applying the scaling  $u^{(phys)} = u^{(LB)} \sqrt{3} c_s^{(phys)}$ .

The local equilibrium is typically a second-order expansion in the fluid velocity of a local Maxwell distribution,

$$f_i^{eq} = w_i \left[ \rho + 3\vec{c}_i \cdot \vec{v} - \frac{3}{2} \vec{v}^2 + \frac{9}{2} (\vec{c}_i \cdot \vec{v})^2 \right], \quad (4)$$

where  $w_i$  is a set of weights normalized to unity. The single-relaxation-time (SRT) LBM, (3), recovers the weakly-compressible, athermal Navier-Stokes equations at low Mach numbers ( $Ma < 0.3$ ) with second order accuracy in space and time.

Nevertheless, the SRT-LBM method shows instabilities when the viscosity is reduced to small values, in order to reach high Reynolds numbers at low Mach numbers. To enhance the stability, the multiple-relaxation-times (MRT) collision operator was proposed by [2, 3]. Instead of relaxing the particle distribution functions themselves towards equilibrium, as in the SRT-LBM, in the MRT-LBM, they are transformed from velocity space into the corresponding moment space, where the moments are relaxed towards their equilibrium values. The moment space of the D2Q9 model has nine velocity moments. The conserved moments are the density (1) and the flow momentum

(2), the non-conserved moments include the energy, the stress tensor components, the energy square and the energy fluxes, for which different relaxation time scales are specified in order to decouple physical from higher order moments, thus improving the numerical stability. The post collision particle distributions  $f_i^{new}$  of the MRT-LBM are then given by the following expression:

$$f_i^{new} = f_i + M^{-1} S (m_i - m_i^{eq}), \quad (5)$$

where  $M$  is the orthogonal transformation matrix,  $m_i = M f_i$  are the moments of the system and  $S$  is a diagonal matrix of the relaxation rates.

A specific MRT variant, adopted in this work, is the Cascaded-Digital-Lattice-Boltzmann (CDLB) algorithm [4], that allows virtually any viscosity value without loss of stability within the low Mach number limit. It adopts central moments in the reference frame moving with the macroscopic velocity and a generalized local equilibrium which is a function of both conserved and non-conserved hydrodynamic moments. The post collision distributions  $\vec{f}^{new}$  of the CDLB are given by

$$\vec{f}^{new} = \vec{f}^{old} + K \cdot \vec{k} \quad (6)$$

where  $\vec{k}$  is the CDLB collision term and  $K$  is the orthogonal transformation matrix, that maps the moments into velocity space.

$$K = \begin{bmatrix} 1 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 4 \\ 1 & 1 & 0 & -1 & 1 & 0 & 0 & 2 & -2 \\ 1 & 0 & 1 & -1 & -1 & 0 & 2 & 0 & -2 \\ 1 & -1 & 0 & -1 & 1 & 0 & 0 & -2 & -2 \\ 1 & 0 & -1 & -1 & -1 & 0 & -2 & 0 & -2 \\ 1 & 1 & 1 & 2 & 0 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 2 & 0 & 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 2 & 0 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & 2 & 0 & 1 & 1 & -1 & 1 \end{bmatrix} \quad (7)$$

As the collision term of the CDLB is rather complex, we refer to the original paper [4].

Using vector notation

$$K = [\vec{K}_0, \dots, \vec{K}_8], \quad (8)$$

$$\vec{f} = \begin{pmatrix} f_0 \\ \vdots \\ f_8 \end{pmatrix}, \quad (9)$$

the conserved moments are expressed as

$$\begin{aligned}
 \rho &= \vec{f} \cdot \vec{K}_0, \\
 \rho v_x &= \vec{f} \cdot \vec{K}_1, \\
 \rho v_y &= \vec{f} \cdot \vec{K}_2.
 \end{aligned}
 \tag{10}$$

### 3 Implementation

Lattice Boltzmann collision equations are usually written in terms of post-collision distributions  $f_i^c$ , i.e.,  $f_i(\vec{x} + \vec{c}_i, t + 1) = f_i^c(\vec{x}, t)$ .

In this contribution a formulation where the pre-collision distribution is described in terms of the post-collision distribution at the respective neighbor node, i.e.,  $f_i(\vec{x}, t) = \mathbf{pre}(f_i(\vec{x}, t)) \equiv f_i^c(\vec{x} - \vec{c}_i, t - 1)$  is used. The operator **pre** of Modelica makes this formulation convenient [5].

#### 3.1 Node element

All elements extend from a basic node element, the partial node model `PrtlNode_D2Q9`. There, all parameters and variables like the transformation matrix, weighting parameters etc. are defined. Furthermore the internal particle distribution variables are defined and initialization values are calculated. Collision and streaming of particles is based on an equidistant time step which is realized by a clock signal

```
clock := sample(dt, dt);
```

where `dt` is the width of one time step, taken equal to one (in lattice units). The  $f_i$  are time discrete quantities changing their values only at event instants which are triggered by the clock signal.

Each grid element has a rest particle and eight particles that are streamed to the first (horizontal and vertical) and second (diagonal) neighbors of the element. To link the elements, two kinds of connectors are implemented, a forward connector `f_fwd` and a backward connector `f_bwd`. Each consists of a Real input variable and a Real output variable to match its counterpart.

```
connector f_fwd
  input Real f_n;
  output Real f_p;
end f_fwd;
```

```
connector f_bwd
  output Real f_n;
  input Real f_p;
end f_bwd;
```

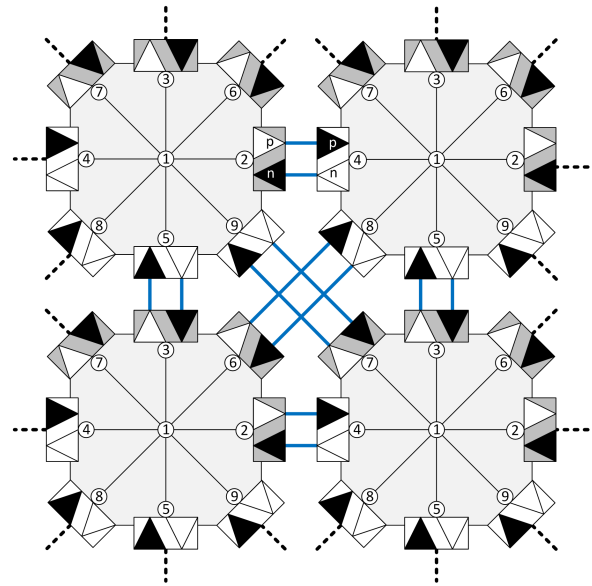


Figure 2: Schematic grid of four D2Q9 node elements including connections

Connectors are placed on the element models facing in all eight streaming directions. Connectors 2, 3, 6 and 7 are facing forwards and connectors 4, 5, 8 and 9 are facing backwards.

Various kinds of node elements extend from this partial model. They are explained in more detail in section 4.

#### 3.2 Collision and streaming step

At every time step and at each grid element, the particle distributions are received, collided and propagated. The receiving and propagating step are generally known as streaming.

In a conventionally implemented LBM, streaming affects only ports of the same direction. This means, a particle distribution with velocity  $\vec{c}_2$  exits port 2 and enters port 2 of the adjacent element. In Modelica, due to the connector concept, the case is slightly different. Port 2 of a grid element is connected to port 4 of the adjacent element, port 4 is connected to port 2 of the next element, and so on. To establish a correct streaming behavior the input, which holds the post-collision value at the last time step, must be mapped to the respective output, e.g., port 4 has to be mapped to port 2, port 2 has to be mapped to port 4. Then the particle distributions are collided. Then the post-collision values are written into the output variable of the respective connector. Propagation to the neighboring grid elements is done automatically by the connector, no additional commands are needed.

```

when clock then
  // mapping
  fold[1] := pre(f1);
  fold[2] := pre(f4.f_p);
  fold[3] := pre(f5.f_p);
  fold[4] := pre(f2.f_n);
  ...

  // collision
  fnew := ... fold;

  // output
  f1 := fnew[1];
  f2.f_p := fnew[2];
  f3.f_p := fnew[3];
  f4.f_n := fnew[4];
  ...
end when;

```

### 3.3 Mesh and connections – setting up the computational domain

In the example model, the 2D-flow model has to be described and the computational domain set up. Each model consists of sources, fluid nodes and boundary conditions. Two ways are possible to build up the model. The first is to build the model by dragging and dropping elements to the workspace and drawing connections by hand. Because the number of elements may be quite high and every element needs eight connects to its neighbours, the effort to set the model up like this is quite high. A more convenient method is proposed here. Providing the matrix `nodeType` that represents the LB discretized computational domain, all connections are generated automatically via nested loops.

The matrix can easily be set up in e.g. Microsoft Excel and then imported to the simulation example. A simple example of a two-dimensional duct model is shown below.

```

parameter Integer nodeType[:, :] =
  {{2, 2, 2, 2, 2, 2, 2, 2},
   {3, 1, 1, 1, 1, 1, 1, 4},
   {3, 1, 1, 1, 1, 1, 1, 4},
   {3, 1, 1, 1, 1, 1, 1, 4},
   {3, 1, 1, 1, 1, 1, 1, 4},
   {3, 1, 1, 1, 1, 1, 1, 4},
   {2, 2, 2, 2, 2, 2, 2, 2}};
...
CDLB.D2Q9 node[:, :] (nodeType=nodeType, ...);

```

The parameter `nodeType` is then propagated to the element `CDLB.D2Q9` which acts as generalized element representing all node types in conditional definition.

```

model D2Q9
  ...
  CDLB.FluidNode   fn if nodeType == 1;

```

```

CDLB.BounceBackNode bn if nodeType == 2;
CDLB.VelocityNode   vn if nodeType == 3;
CDLB.DensityNode    dn if nodeType == 4;
...
end model;

```

The user only has to define the matrix, all connections are established automatically. They are defined in multiple loops to interconnect every element with its eight neighbors. As example, the connections for connector 4 are outlined here:

```

// connect row 2:end and col 2:end
for i in 2:1:nrow loop
  for j in 2:1:ncol loop
    connect (node[i, j].f4, node[i, j-1].f2);
  end for;
end for;

// connect row 1 and col 2:end
for j in 2:1:ncol loop
  connect (node[1, j].f4, node[1, j-1].f2);
end for;

// connect col 1 to col end
for i in 1:1:nrow loop
  connect (node[i, 1].f4, node[i, end].f2);
end for;

```

These equations are repeated for all other connectors and are omitted here for sake of brevity.

## 4 Elements

### 4.1 FluidElement

The fluid element implements the collision, (6). To speedup symbolic pre-processing and compilation time, this is encapsulated in a function. To avoid reflections at the in- and outflow boundaries, sponge zones are implemented. There the relaxation factor  $\tau$  is gradually increased to 1, thus driving the fluid towards its equilibrium state at the boundary.

### 4.2 BoundaryElement

To implement a solid, non-slip boundary condition a local bounce-back rule is applied on the solid node  $\vec{x}_s$ :

$$f_{i'}^c(\vec{x}_s, t) = f_i(\vec{x}_s, t) \quad (11)$$

where  $i'$  denotes the link with reversed velocity of  $i$ , i.e.,  $\vec{c}_{i'} = -\vec{c}_i$ , pointing into the fluid. Incoming distributions functions at a wall node are reflected back to the original fluid nodes, with the direction rotated by  $\pi$ . The “bounce-back on the node” is purely local, thus being implementable in the current concept, but it has



been proven to be only first-order accurate in time and space.

Shifting the solid wall half-way between the two nodes, leads to the “bounce-back on the link” which is of second order accuracy:

$$f_i'(\vec{x}_l, t+1) = f_i^c(\vec{x}_l, t) \quad (12)$$

where  $\vec{x}_l$  is a fluid node next to the solid boundary and  $f_i^c$  is the post-collision value before propagation. Unfortunately it is not implementable in the current context, as each node element in Modelica cannot access its neighbors.

### 4.3 DirichletElement

Dirichlet boundary conditions can be set up based on the idea of bounce-back of the non-equilibrium part, as proposed in [6]. As an example, at a flow boundary having a normal vector into the fluid in positive  $x$ -direction, i.e.,  $f_2, f_6, f_9$  pointing into the fluid, these distribution functions are unknown after streaming. Equations (1) and (2) can be used to reconstruct the unknown distributions.

At a velocity boundary node, after streaming  $f_1, f_3, f_4, f_5, f_7, f_8$  are known and  $v_x, v_y$  are specified.  $f_2, f_6, f_9$  and  $\rho$  need to be determined. Equations (1) and (2) yield three equations. In order to close the system, it is assumed that it is admissible to bounce-back of the non-equilibrium part of the particle distribution normal to the boundary, i.e.,  $f_i^{neq} = f_i^{neq} \equiv f_i' - f_i^{eq} = f_i - f_i^{eq}$ . For a Dirichlet element pointing in positive  $x$ -direction this gives

```
rho := 1/(1-vx) * ((fo[1]+fo[3]+fo[5]) +
  + 2 * (fo[4]+fo[7]+fo[8]));
fo[2] := fo[4] + 2/3*rho*vx;
fo[6] := fo[8] + 1/2*(fo[5] - fo[3])
  + 1/2*rho*vy + 1/6*rho*vx;
fo[9] := fo[7] + 1/2*(fo[3] - fo[5])
  - 1/2*rho*vy + 1/6*rho*vx;
```

For a known inlet velocity  $v_{in,x}$ , this system serves as velocity inlet, `CDLB.VelocityNode`, or can be rearranged in terms of known density,  $\rho = \rho_{in}$ , into a density inlet, `CDLB.DensityNode`.

### 4.4 Initial Conditions

At start of the simulation run, a flow at rest is assumed, setting the distribution functions to their equilibrium value.

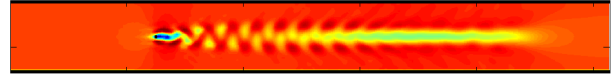


Figure 3: Profile of the magnitude of the flow velocity in lattice units in a fluid domain with one solid node at one quarter of the domain length and vertically centered. A periodic vortex occurs forming a von Karman vortex.

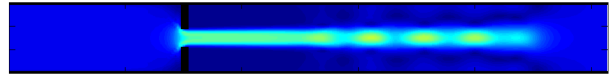


Figure 4: Profile of the magnitude of the flow velocity in lattice units in a fluid domain with an orifice at one quarter of the domain length. A jet is created which is amplified by aerodynamic effects.

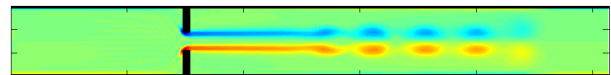


Figure 5: Profile of the vorticity in lattice units in a fluid domain with an orifice at one quarter of the domain length. A jet is created which is amplified by aerodynamic effects.

## 5 Test cases and results

### 5.1 Flow past a cylinder

The first test case deals with the flow around a cylinder in a fluid stream, which is formed by one solid node. It is positioned at approximately one quarter of the domain length and vertically centered in the computational domain. The nodes at the upper and lower boundaries are also fluid nodes. As all boundaries are interconnected with each other, this constitutes periodic flow boundary conditions. The computational domain consists of  $258 \times 30$  grid nodes, leading to 570k equations, which is close to the maximum size manageable in Dymola.

Periodic vortex at the cylinder occurs forming a von Karman vortex street, see Figure 3 for a snapshot showing the velocity magnitude in lattice units. In order to obtain a high Reynolds number, the relaxation factor  $\tau$  was set to  $1/2$ , yielding a very low kinematic viscosity, which is only determined by numerical precision of the solver.

### 5.2 Flow through an orifice

The second test case is a flow through an orifice where a jet is formed. The orifice is modeled as solid with

non-slip walls and is positioned in a duct with slip-walls at one quarter of the domain length. The computational domain consists of 258 x 30 grid nodes, leading to 570k equations.

Figures 4 and 5 show snapshots of the velocity magnitude and the vorticity in lattice units. Velocity perturbations at the flow inlet trigger initial perturbations in the jet shear layers which are further amplified by aerodynamic effects due to the Kelvin-Helmholtz instabilities until the jet breaks up into discrete vortices.

## 6 Conclusions

An approach for simulating fluid flow with the Cascaded Digital Lattice Boltzmann method is proposed. With this approach fluid flow problems can be addressed in a multi physical modeling language like Modelica. The method works for small scenarios but reaches the boundaries of efficient simulation quickly. Nevertheless, the approach works for 2D flows and can be extended to 3D flows easily.

## References

- [1] M. Bonvini, M.; Popovac, “Fluid flow modelling with Modelica,” *Proceedings of the 7th International Conference on Mathematical Modelling, Vienna, Austria, 2012*.
- [2] D. D’Humières, “Generalized lattice-Boltzmann equations,” *Rarefied Gas Dynamics: Theory and Simulations*, vol. 159, pp. 450–458, 1992.
- [3] P. Lallemand and L.-S. Luo, “Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability,” *Phys. Rev. E*, vol. 61, pp. 6546–6562, Jun 2000.
- [4] M. Geier, J. G. Korvink, and A. Greiner, “Cascaded digital lattice Boltzmann automata for high Reynolds number flow,” *PHYSICAL REVIEW E* 73, 2006.
- [5] J. Brown, “Computational fluid dynamics in an equation-based, acausal modeling environment,” Master’s thesis, Atlanta, Ga. : Georgia Institute of Technology, 2010.
- [6] Q. Zou and X. He, “On pressure and velocity boundary conditions for the lattice Boltzmann BGK model,” *Physics of Fluids*, vol. 9, no. 6, pp. 1591–1598, Jun. 1997.

# FORM-L: A MODELICA Extension for Properties Modelling Illustrated on a Practical Example

Thuy Nguyen  
EDF R&D

6, quai Watier F-78401 Chatou Cedex, France

## Abstract

As systems engineering methodologies for complex systems make increasing use of modelling and simulation techniques, it has become important to extend the MODELICA language to also cover requirements, and more generally, properties modelling. The ITEA2 MODRIO project is currently developing an extension for that very purpose: the FORM-L language (FOrmal Requirements Modelling Language). This paper presents an overview of the FORM-L concepts, and illustrates them with examples based on a practical case study, the Backup Power Supply (BPS) system.

*Keywords: physical modelling; requirement modelling; systems engineering; methodology*

## 1 Introduction

Systems engineering methodologies for complex systems increasingly rely on, or could benefit from, modelling and simulation. For MODELICA to support activities such as functional validation of system requirements, design verification against requirements, testing, dysfunctional analyses and verification of operational procedures, the ITEA2 MODRIO project is developing extensions to the language. One of them concerns formal requirements and properties modelling, and is called FORM-L (FOrmal Requirements Modelling Language). This paper presents the main concepts underlying FORM-L, and illustrates them with examples taken from a MODRIO case study, the Backup Poser Supply (BPS) system.

Section 2 presents the main objectives assigned to FORM-L. Section 3 introduces briefly the BPS case study in order to provide a background context for the examples given in the following sections. Section 4 presents how FORM-L considers *functions*, *constants* and *fixed* variables. Section 5 introduces the notions of *condition* and *event*. Section 6 presents the

notions of *properties*, *requirements*, *assumptions* and *guards*. Section 7 presents the notion of *time locator*, continuous or discrete. Section 8 presents how FORM-L views *sets* and *arrays*. Lastly, Section 9 presents how *actions* are modelled in FORM-L.

## 2 FORM-L Overview

### 2.1 Motivation

Paper *Innovative Modelling Architecture for the Verification of Design against System requirements* presents how one of the methodologies developed by MODRIO (to verify the design of a system against its requirements) is supported by FORM-L.

This includes in particular a clear separation of models serving different purposes in the systems engineering lifecycle or the support to systems operation. In particular, there should be a well-identified model that clearly and formally specifies:

- The boundaries of the system under study.
- The interactions of the system with its environment (including human operators), including any assumptions made regarding this environment.
- The system requirements, including functional and timing requirements (concerning the interactions with the environment) and system operational requirements (including quality of service and fault-tolerance, and operational constraints aiming for example at reducing wear and tear of system components).

### 2.2 Main Notions

This is supported by the FORM-L with the notions of *property*, *requirement*, *assumption*, and *external* information (to be supplied either by other MODELICA models or by engineering databases).

The expression of a requirement or a desirable property needs to address four basic issues: WHAT, WHERE, WHEN (cf. EuroSysLib WP7.1 Property Modelling) and HOW WELL.

WHAT states what needs to be achieved or what must be avoided. This is expressed in FORM-L with the notions of *condition* (that must be satisfied), *event* (that must or must not occur), *function* and *action*.

WHERE states where in the system the WHAT needs to be achieved. This can be expressed in FORM-L by the explicit naming of objects, but also with the notion of *set*, in particular of set resulting from queries. Indeed, at the time system requirements are specified, early in the system lifecycle, the names, number, types, characteristics and locations of the objects concerned are often not known yet. They are determined at a later stage, the information being usually stored in one or more engineering databases.

WHEN states when the WHAT needs to be achieved. Possibly complex temporal logic is often needed when considering reactive systems such as the BPS. Such logic can be expressed in FORM-L using *continuous* or *discrete time locators*. FORM-L also includes the notions of *finite state automaton*, *statechart* and *time domain* (the latter being extremely useful, or even necessary, when considering hybrid systems).

HOW WELL states how well the WHAT needs to be achieved (as real life systems are bound to have failures). This is expressed in FORM-L using probabilistic properties. Extensive work is being done in the framework of MODRIO on stochastic issues and multi-mode modelling: FORM-L only addresses what concerns properties.

### 2.3 Readability

It is not sure that all aspects of requirements specification (in particular complex temporal logic) can be represented graphically without risks of misinterpretation from the part of readers, or even authors of models. Therefore, the clarity of the FORM-L language textual syntax is important, as the language is mainly intended to be used by application specialists rather than modelling experts. The syntax that is proposed here voluntarily includes significant amounts of "syntactic sugar" for this very reason.

## 3 A Brief Introduction to the BPS Example

### 3.1 BPS Objective

The objective of the BPS (Backup Power Supply) is to provide electric power to electrical components (e.g., pumps and valves in a thermohydraulic indus-

trial installation) that are considered essential, in case of loss of the Main Power Supply (MPS). Such components could be important to safety (e.g., in an industrial installation or in a hospital) or could be required to prevent unacceptable economic losses (in a semiconductor fab).

Figure 1 presents the overall organisation of the BPS from the standpoint of the requirements specifiers. Figure 2 presents a system architecture developed by designers as a possible answer to the requirements. The objective of the system requirements model and of the architecture model is to support the verification that the architecture indeed meets the requirements.

### 3.2 BPS Principles

As the levels of power required by the backed-up sets of components are assumed to be very high, and the duration of the backing-up to last up to several days, the BPS is based on a Back-Up Generator, or BUG. This generator has a number of constraints. In particular, it cannot power instantaneously all necessary components: if all 'client' components were connected simultaneously to the BUG, the BUG could be overloaded and stall, due to the fact that when electric power is restored to a given component, there is an important, transient call for current and power (see figure 3). Thus, much like a conventional car engine needs a gearbox, the BPS and its BUG need an active control system to ensure a progressive and orderly increase of requested power (hence the presence of circuit breakers).

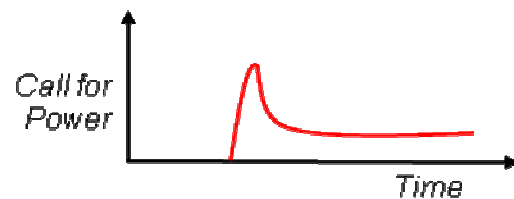


Figure 3: Transient call for current when electric power is supplied to an electric component

Also, not all components powered by the backed-up electric panel have the same needs and functional roles. In the case study, six sets of backed-up electric components (SBC) have been identified:

- *SBC1* groups components that implement Context Specific Actions (CSA). Such actions are not always needed. However, if and when they are, then Set1 must be powered back within 20 seconds.
- *SBC2*, *SBC3*, *SBC4* and *SBC5* are redundant sets of components, and it is sufficient to power any two of them within 40 seconds.

- *SBC6* must be powered back within 60 seconds.

When in operation, the *BPS* can be in one of three main states:

- *Nominal* state: the *BPS* is available and ready to perform its missions.
- *Test* state: as *MPS* loss is rare and the *BPS* is seldom required, periodic testing is necessary to

- ensure that when needed, the *BPS* will indeed be able to perform its missions.
- *Maintenance* state: the *BPS* is under repair and is not able to perform its main mission, which is to provide electric power to the backed-up electric panel.

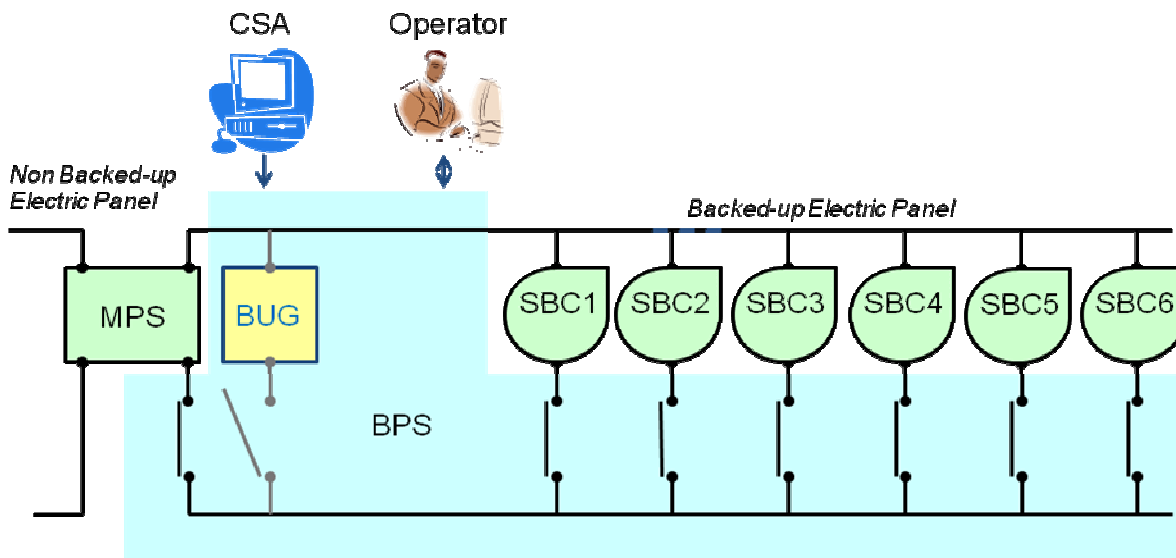


Figure 1: *BPS* and its environment as viewed by system requirements specifiers. The overall configuration corresponds to the case where the *MPS* is available

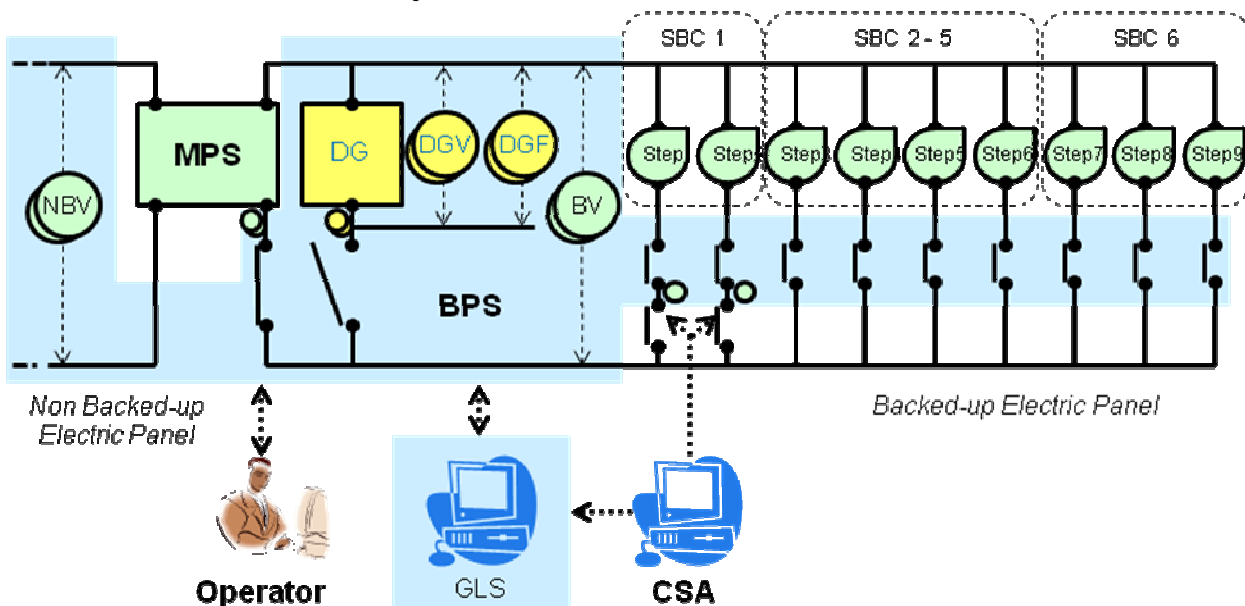


Figure 2: *BPS* and its environment as viewed by the system designers. Note the introduction of sensors (voltmeters, frequency meters, breaker position sensors), of the digital control system *GLS* (Generator Load Sequencer), of the splitting of certain *SBCs* into smaller Steps.

### 3.3 The BPS Models

The BPS case study considers 4 models (see figure 4):

- *BPS.REQ* is a property model specifying the functional and quality of service requirements applicable to the *BPS*. This property model also describes the environment of the *BPS*, and the interactions between the *BPS* and that environment.
- *BPS.ADS* is a property model describing an architectural design for the *BPS*. In particular, it identifies the components constituting the *BPS*, and puts requirements on these. One objective is to verify that this design will indeed satisfy the requirements stated by *BPS.REQ*.
- *BPS.ENV* is a behavioural model that simulates the system environment of the *BPS*, including operators' actions.
- *BPS.BEV* is a behavioural model that simulates the design specified by *BPS.ADS*.

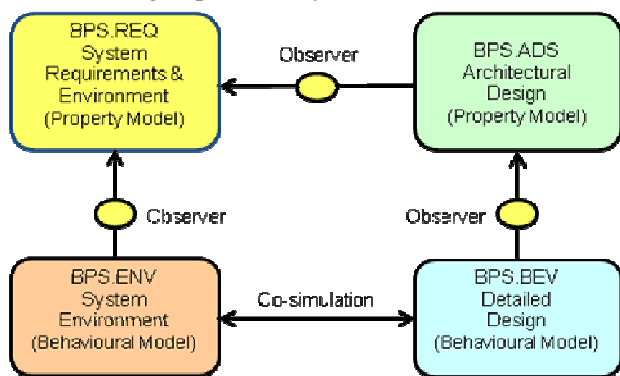


Figure 4: The MODELICA models for the BPS

## 4 Functions, Constants and Fixed Values

### 4.1 Functions

Functions are features the value of which depend on time. In addition to functions already supported by current MODELICA, FORM-L has a few additional types:

- *Conditions* are combinations of Boolean and temporal logic. Their values can be *true*, *false* or *undefined*.
- *Finite state automata* are functions the values of which are in an enumerated set (see section 10 Discrete States - Finite State Automata).
- Probabilities are functions the values of which are real in the  $[0., 1.]$  range (see section 13 Probabilistic Properties).

### 4.2 Constants

Qualifier **constant** may be used to specify that a feature does not vary in time and has the same value for all simulation runs. This is not absolutely necessary (the FORM-L compiler should be able to detect that automatically) but may clarify authors' intentions.

```
constant real pi = 3.1416;
```

### 4.3 Fixed Values

Qualifier **fixed** may be used to specify that a feature has a value that is determined at the beginning of a run and does not vary during that run. However, it may be different in different runs. Here again, this is mainly to clarify authors' intentions.

```
constant duration CycleTime = ms50;
```

```
fixed duration Phase = random (0.0, CycleTime);
```

The GLS (Generator Load Sequencer, see Figure 2) is a digital, synchronous control system introduced by the architectural design. It operates in a discrete time domain, the *CycleTime* of which is 50 milliseconds (ms50). The *Phase* of the time domain is a random value that does not change once the system has started, but that will be different when the system is restarted.

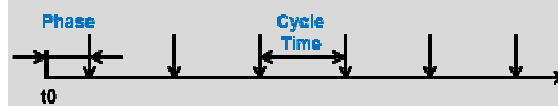


Figure 5: The GLS time domain

## 5 Conditions and Events

### 5.1 Conditions

Conditions determine the evaluation of Boolean expressions restricted to time periods specified by *continuous time locators* (or CTLs). The general rule is that they are *true* when not in the time period, and take the value of the Boolean expression when in the time period. However, there are cases where they are undefined (see example below).

Conditions are used to specify so-called *condition-based properties*. The difference between the two is that properties express something that is desirable, required or assumed, whereas conditions are just building blocks to express properties, as it is often preferable to break the expression of complex properties into simpler, intermediate expressions.

```
condition C = duringAny s4 check Off;
```

This condition is used to specify when the MPS can or must be declared unavailable. In particular, to avoid activating the BPS for short MPS losses, it must have been off for at least 4 consecutive seconds (s4).

Condition *C* is evaluated at the end of each 4 second time window (**duringAny** s4): it is *true* if *Off* had been *true* during the complete time window. It is *false* otherwise. It is *undefined* during the first 4 seconds of the simulation run, since no 4 second time window has elapsed yet.

## 5.2 Events

An *event* characterises the occurrence of one or more facts that have no duration: each of these facts is an occurrence of the event. Events are used to specify so-called event-based properties.

```
external Boolean EndMaint;
event endMaint =
  when EndMaint becomes true;
```

The external Boolean *Endmaint* represents the time-continuous signal issued by one of the buttons at the disposal of the human operator. This signal is simulated by the behavioural model of the BPS environment. Transition from *false* to *true* (**becomes true**) denotes the end of the on-going maintenance operation.

## 6 Properties, Requirements, Assumptions, Guards

There are two types of properties: condition-based properties, and event-based properties. Two Boolean functions are attached to each property:

- *Violated* is initially *false*. It becomes *true* at the first instant where the property is violated, and remains so until the end of the simulation run.
- *Evaluated* is also initially *false*. It becomes *true* at the first instant where the *Violated* / *notViolated* status can no longer be modified in the course of the simulation run, and remains so until the end of the simulation run.

### 6.1 Condition-Based Properties

Like a condition, a condition-based property specifies a Boolean expression and a CTL: the Boolean expression should be *true* during the CTL.

A three-valued function (*satisfied*, *notSatisfied*, *notApplicable*) is attached to each condition-based property. During the CTL, the property is *satisfied*

when the Boolean expression is *true*, and *notSatisfied* otherwise. It is *notApplicable* at time instants not covered by the CTL or when the combination of CTL and Boolean expression is *undefined*. It becomes *Violated* at the first instant where it is *notSatisfied*.

```
property P2ce =
  after (BPSNeeded becomes true)
  within s60
  check SBC[6].Powered becomes true;
```

This property expresses the need to start providing electric power to the 6th SBC (SBC[6].Powered **becomes true**) at most 60 seconds after the BPS has been declared as needed.

### 6.2 Event-Based Properties

An event-based property specifies a constraint on the number of occurrences of an event during a given time locator.

A three-valued function (*belowLimits*, *withinLimits*, *aboveLimits*) is attached to each event-based property, indicating whether the number of occurrences is below, within or above the limits specified. The property becomes *Violated* at the first instant where it is *aboveLimits*.

```
property P2 = {P2a; P2b; P2c};
required property R7 =
  until (or{P in P2 | P.Violated})
  becomes true
  check no eFailure;
```

P2 is the set of properties stating when SBC1 (P2a), SBC2-5 (P2b) and SBC6 (P2c) must be powered by the BPS. When any one of the SBCs cannot be powered back within the allocated time, a failure signal (eFailure) must be sent to the operator.

Property R7 expresses the requirement that this signal must not be sent (**no eFailure**) spuriously, i.e., before any one of the P2 properties is violated.

### 6.3 Requirements,

A *requirement* is a property that **MUST** be satisfied: it is the objective of simulation to verify that it is not *Violated*.

### 6.4 Assumptions

An *assumption* is a property that is supposed to be satisfied: simulation scenarios assume / ensure that it is satisfied. Assumptions are usually made with respect to the environment or the boundaries of the system under study. They can also be made on sys-

tem aspects not fully determined yet, in preliminary stages of design.

```

assumed property
  during MPS.Unavailable
  check no eMaint;
  
```

This assumption concerns the human operator's actions. It states that when the MPS is not available, the operator will not launch a maintenance session (**no** eMaint).

### 6.5 Guards

A guard is a property that states the conditions that must be satisfied for a model to be valid. It is to be used for multi-modelling, when several models are available for the same system or components, each corresponding to specific situations.

## 7 Time Locators

There are two types of time locators in FORM-L: continuous time locators (CTLs) and discrete time locators (DTLs).

### 7.1 Basic Continuous Time Locators (CTLs)

A CTL specifies one or more time periods. Time periods have a duration and usually have a position in time (see Figure 6).

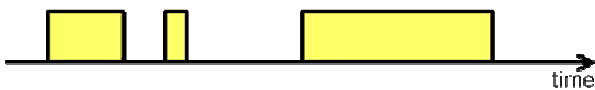


Figure 6: Time intervals

#### duringAny duration

This defines a sliding time window, i.e., any time period of a given duration (see Figure 7 and example in Section 5.1).



Figure 7: *duringAny* duration

Sliding time windows are a very distinct type of CTL and cannot always be used as the other types of CTL.

#### during condition

The time periods defined are those where condition is **true** (see figure 8).

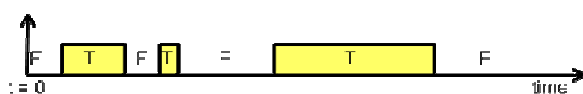


Figure 8: *during* condition

```

property P4 =
  during not (BPSNeeded) // CTL
  check not (Active);
  
```

In this example, the CTL is specified in line 2. The property states that when the BPS is not needed, it should not be activate.

#### after event

The time periods defined begin with each occurrence of event and last until the end of the simulation run (see figure 9). There are as many periods as there are event occurrences: if there are more than one occurrence, they will overlap.

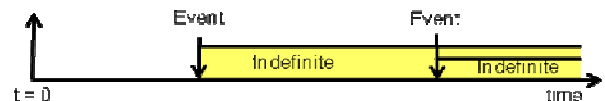


Figure 9: *after event*

```

event eMustAbortM =
  (after eMaint within s60)
  and MPS.eLoss;
  
```

In this example, the meanings of the named events and conditions are as follows:

- eMustAbortM is an event that signals that a BPS maintenance request previously raised by the operator needs to be cancelled.
- eMaint is an event raised by the operator signalling that maintenance will be performed on the BPS.
- MPS.eLoss is an event signalling that the MPS has been lost, and thus that the BPS is needed.

This event occurs when the MPS is lost 60 seconds or less after a maintenance request has been issued. In such cases, the maintenance request is expected to be aborted (see example below).

#### after event for duration

#### after event within duration

The time periods defined begin with each occurrence of event and last for the specified duration (see figure 10).

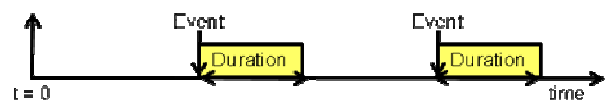


Figure 10: *after event for duration*

Both syntaxes have the same meaning, but whereas the expression with **for** is mainly used for condition-based properties (the condition must be true **for** a certain time period), the expression with **within** is mainly intended for event-based properties (an event must occur **within** a certain time period).



```

assumed property
  after eMustAbortM within s5
  check endMaint;
  
```

This statement assumes that within 5 seconds ( $s5$ ) after an occurrence of `eMustAbortM` (the event signalling that a BPS maintenance request should be cancelled), the effective cancellation event (`endMaint`) is indeed raised by the operator.

**after event1 untilNext event2**

The time periods defined begin with each occurrence of `event1` and last until the first strictly following occurrence of `event2` (see figure 11). There are as many intervals as there are occurrences of `event1`.

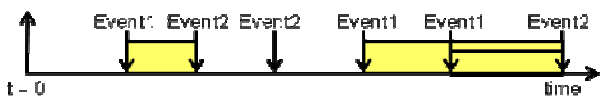


Figure 11: *after event1 untilNext event2*

```

condition Running =
  after eStart untilNext eStop;
  
```

This statement concerns the backup generator of the BPS. The generator can signal several events, including `eStart` (it has started) and `eStop` (it has stopped). The statement defines the time periods where the generator is Running.

**until event**

The time intervals defined all start at the beginning of the simulation run. There is one time interval per occurrence of `event`, and it ends with the occurrence (see figure 12).

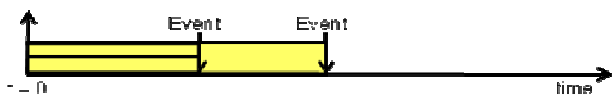


Figure 12: *until event*

```

condition BPSNeeded =
  (MPS.Unavailable or after Op.eVTest)
  and not(Maintenance)
  and until Op.eVReset;
  
```

This statement specifies when the BPS is needed. It is a time period that begins when the MPS has become unavailable (`MPS.Unavailable`) or when the operator has initiated a periodic test (`Op.eVTest`), under the condition that the `BPS` is not under maintenance (`not(Maintenance)`). It ends when the operator issues a valid reset (`Op.eVReset`):.

**every duration1 for duration2**

This expression defines periodic time intervals (see figure 13). `duration2` should be shorter than `duration1`.



Figure 13: *every Duration1 for Duration2*

```

required property
  every s10 for s2 check eCheck;
  
```

This statement requires that periodically, every 10 seconds ( $s10$ ), event `eCheck` should occur in the first 2 seconds ( $s2$ ).

### 7.2 Combining / Transforming CTLs

The FORM-L offers various means for deriving new CTLs from already existing ones. These are summarised in figure 14. The only type of CTL that cannot be used in these expressions are sliding time windows: they can be neither transformed nor combined directly, but conditions based on sliding time windows can be used (with keyword `when`) to define CTLs that can be combined and transformed.

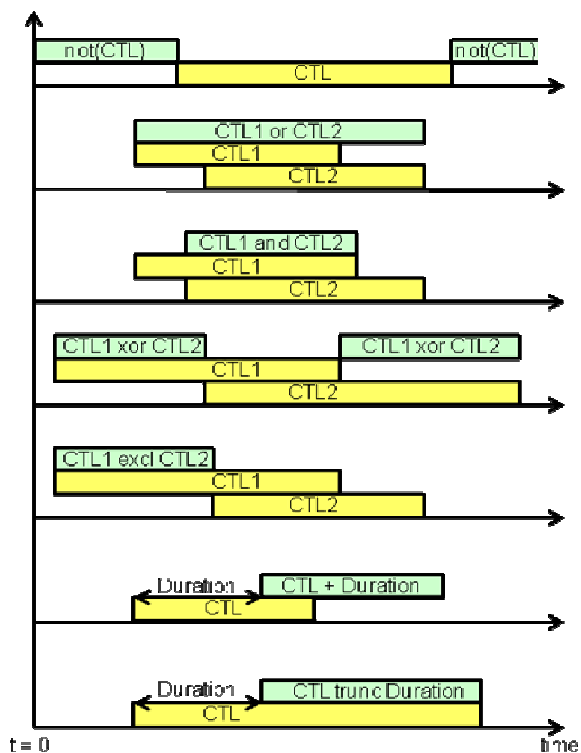


Figure 14: *Deriving new CTLs from existing ones*

### 7.3 Basic Discrete Time Locators (DTLs)

A DTL defines one or more positions in time and has no notion of duration.

- when condition becomes true
- when condition becomes false
- when condition changes

The DTL has a time position for each instant where condition becomes *true*, *false* or changes value (see Figures 15 and example of Section 6.1). In a discrete time domain, condition changes value when and only when its value at a given instant is different from its value at the preceding instant.

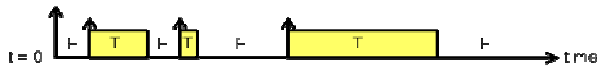


Figure 15a: when condition becomes true



Figure 15b: when condition becomes false

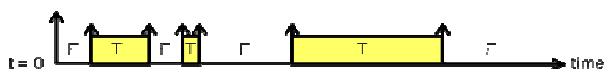


Figure 15c: when condition changes

- when fsa becomes state
- when fsa leaves state
- when fsa changes
- when integer changes

DTLs can be defined when a finite state automation (fsa) enters or leaves a given discrete state or changes state. They can also be defined when an integer function changes value.

when event

This expression simply denotes the DTL associated with an event.

every duration

This expression defines a periodic DTL, the period of which is duration (see figure 16).



Figure 16: every duration

```
constant duration CycleTime = ms50;
private fixed duration Phase =
  random(0.0, CycleTime);
dt1 = (every CycleTime) + Phase;
```

This example defines the discrete time domain associated with the GLS, which has a synchronous design: it sees its environment and acts upon it only at the instants defined by the specified dt1. The period

is defined by CycleTime (50 milliseconds). The time domain has a random phase defined by Phase.

7.4 Combining / Transforming DTLs

FORM-L offers various means for deriving new DTLs from already existing ones. These are summarised in figure 17.

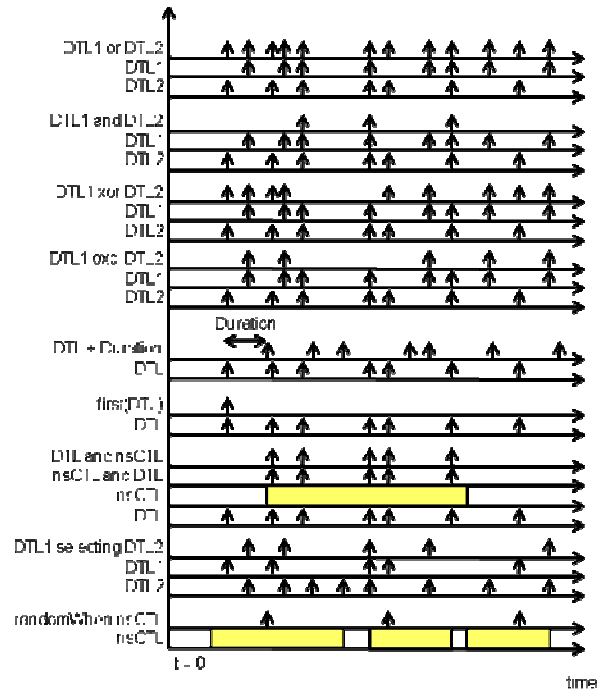


Figure 17: Deriving new DTLs from existing ones or from CTLs

8 Sets, Subsets and Arrays

Sets are an essential ingredient of FORM-L, particularly when requirements and assumptions are stated at very early stages of a project, where the precise identifications and numbers of the objects constituting the system are yet unknown.

Sets are first divided into two main categories: *static sets* (the membership of which does not vary in time) and *dynamic sets*.

Static sets are either *enumerated sets* (their members are listed individually) or *queried sets* (their membership is defined in intention based on constraints on static attributes, and is obtained at the beginning of a simulation run through a query to an engineering database).

Dynamic sets are always subsets of static sets. Their membership is defined based on constraints involving dynamic attributes.

All sets are ordered, either explicitly or implicitly. In this sense, they are close to single dimension arrays (vectors) of the current MODELICA language.

However, there is one significant difference: an array contains its members and an object cannot belong to two different arrays; a set references its members, and an object may belong to several sets.

FORM-L has set operators (such as union or intersection), arithmetic operators (such as sum, minimum or maximum) and Boolean operators (such as and or or). It also provide a cardinal function that calculates the number of members in a set and that allows the expression of universal or existential quantifiers.

## 9 Actions

FORM-L provides the notion of action to support the modelling of designs.

### 9.1 Elementary Actions

Elementary actions belong to one of two types:

- Assignments to functions.
- Raising of events.

They are composed of two parts: an optional delay part (specified by a time locator) and an action specification (the effective event raising or function assignment).

### 9.2 Composite Actions

Composite actions regroup two or more actions that need to be performed in a coordinated manner at instants specified by a CTL or during time periods specified by a DTL. There are two types of coordination:

- Sequences, where the member actions are performed one after the other.
- Simultaneous actions, where all actions are performed at the instants specified by a DTL.
- Non-ordered actions, where the actions are performed at unspecified instants within the time periods specified by a CTL.

```
when State becomes Operational(Active;)
then sequence
  raise eStartDG;
  wait eDGReady
  then raise eOpenMPSBrk;
  wait eMPSBrkOpen
  then raise eShedAll;
do
  wait s1 then raise eCloseDGBrk;
  wait s5 then raise eReload;
end;
end;
```

This composite action specifies the elementary actions to be implemented by the GLS when the BPS is needed (see Figure 17).

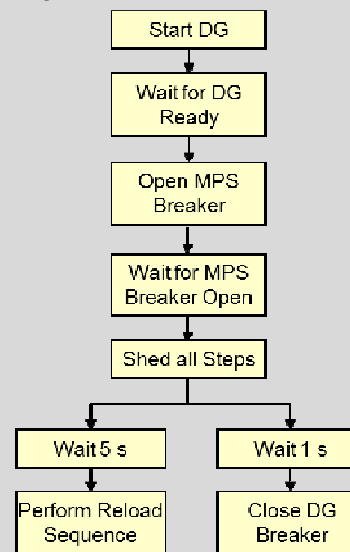


Figure 17: Beginning of the GLS sequencing

## 10 Probabilistic Properties

The MODRIO project is developing extensive stochastic modelling capabilities for MODELICA. These will not be addressed here: for requirements modelling, FORM-L needs only two simple notions.

### 10.1 Probability Function

The first notion introduced by FORM-L is a function of time: its value at a given instant is a Real number in the  $[0., 1.]$  range that represents the probability that a specified event occurred at least once before that instant.

```
property P3 =
  during not(BPSNeeded)
  check not(Active);

required property R3 =
  when SingleSensorFailure then P3;

required property R9 =
  probabilityFunction
  (P3.Violated becomes true)
  < 1-exp(-Y*time);
```

Property P3 states that the BPS should not be spuriously activated when it is not needed. Requirement R3 states that when there are no BPS component failures, or at most one sensor failure, then P3 must be satisfied. Requirement R9 puts a limit on the probability of spurious activation of the BPS (due to BPS components failures). This probability must be lower than what it would be if there was a constant occurrence rate of spurious activation, equal to Y.

## 10.2 Probability

The second second notion introduced by FORM-L is a Real constant or fixed value (not a function of time) in the  $[0., 1.]$  range that states the probability of an event occurring at least once during a given time period (possibly during the whole simulation run). This could be used for example to specify conditional probabilities, with the time period representing the condition. Application of conditional probabilities include probabilities of failure on demand or probabilities of common-cause failure.

```
property P2ce =
  after (BPSNeeded becomes true)
  within s60
  check SBC[6].Powered becomes true;
property P2e = {P2ae; P2be; P2ce};
required property R3a =
  probability (
    (card{P in P2e | P.Violated} == 1)
    becomes true)
    < 5*10-3;
```

Property `P2ce` states that `SBC6` (`SBC[6]`) should start to be powered within 60 seconds after the BPS is declared needed. There are similar properties for the other SBCs (`P2ae` and `P2be`). They are grouped in `P2e` which is the set of properties regarding the timeliness of providing power to the SBCs.

Requirement `R3a` specifies the maximum probability of not satisfying any single one of these properties.

## 11 Conclusions

A number of tasks remains to be achieved, mainly:

- Finalizing the FORM-L concrete syntax, in order to bring it closer to the current MODELICA syntax when this is not at the cost of clarity and conciseness.
- Deciding on an implementation path in the existing tools environments. It is likely that integration with other existing tool environments (in particular with Computer Aided Design and Product Life Management environments and their data bases).
- Developing the methodological aspects to better support systems engineering and systems operation activities.

## 12 Acknowledgments

This work is partially supported by DGCIS within the ITEA 2 MODRIO project.

## References

- [1] Schamai W., Buffoni L., Bouskela D., Fritzson P., 'Automatic Model Composition using Bindings in Modelica', Modelica 2014 conference proceedings, Lund, 2014.
- [2] Bouskela D., El Hefni B., 'A physical solution for solving the zero-flow singularity in static thermalhydraulics mixing models', Modelica 2014 conference proceedings, Lund, 2014.
- [3] Bouskela D., Jardin A., Nguyen T. 'Innovative Modelling Architecture for Design Verification against System requirements', , Modelica 2014 conference proceedings, Lund, 2014.

# Statecharts as a Means to Control Plant Models in LMS Imagine.Lab AMESim

Vincent Berthoux Sébastien Furic Loïc Wagner  
{vincent.berthoux,sebastien.furic,loic.wagner}@lmsintl.com  
LMS Imagine S.A.  
7 place des Minimes  
42300 Roanne, France

## Abstract

This article introduces a new feature of LMS Imagine.Lab AMESim that allows users to define plant model controllers. We start by reviewing some challenging aspects of hybrid state machine handling in asynchronous Modelica-based physical simulation environments. We then describe the implementation available in AMESim, focusing on user interaction and especially static error checking and reporting.

*Keywords:* Statechart; Modelica; LMS Imagine.Lab AMESim

## 1 Background

Models of physical systems can be built out of equation-based entities (submodels) whose interaction through a connection structure yield the behavior under consideration. This way of defining physical models puts emphasis on *technological* and/or *phenomenological* aspects of modeling: one typically defines entities representing fundamental phenomena (e.g., energy storage), or entities representing technological assemblies (e.g., a cooling system), or any combination of both, as modeling ‘bricks’. However, this is not the only nor always the most appropriate way of defining models. For instance, one sometimes prefers to put emphasis on *states* and *transitions* between states. This is typically the case when building controllers used to drive models. These controllers feature operating modes that can be conveniently represented as states of a certain finite state machine (consider for instance a controller having modes *start*, *run* and *stop* with transitions between these states indicating possible mode transitions).

To make the picture complete however, real-world controllers actually also feature *state variables*, leading to infinite (often uncountable) hybrid state ma-

chines. Nevertheless, the state-and-transition view is still the preferred one in most situations: this observation motivated the introduction of a new user interface feature in AMESim, allowing users to define controllers by means of a finite set of states and transitions, yet offering state variables and equations as a means to specify not only actions to be performed during state transition but also constraints to be verified in a given state.

Figure 1 shows a simple counter model expressed in the new state-and-transition view, which has been highly inspired by Harel’s statechart language [1]. The corresponding user interface has been built on top of AMESim’s Modelica translation chain to benefit from its automatic code generation feature.

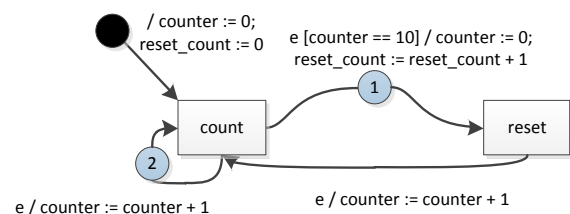


Figure 1: A simple counter statechart

## 2 Why statecharts?

Variants of Harel’s statecharts are today by all means one of the most popular approaches used to describe state machines in Control tools. On the other hand, LMS Imagine.Lab AMESim, which is a Physical System Modeling tool, used to favor the technological and phenomenological aspects of models—which are often the most natural ones in its application area. However, models do not necessarily classify as “pure control” or “pure physical”: some of them involve a *mix*

of physical and control aspects (e.g., aircraft models including aircraft missions, vehicle models including driving manoeuvres, etc.). If we want to handle such models in a simulation tool, we basically have two options: either we choose a unified representation, or we offer the ability to work with both the state-and-transition view and the technological-and-phenomenological view—and then possibly translate heterogeneous parts to a common representation under the hood. In the following we explain the reasons that have driven the choice made in AMESim, whose last release implements the second option.

## 2.1 Control in AMESim by means of native components

Early attempts to mix control with physics in AMESim naturally made use of the versatile native *component* concept. An AMESim component can be seen as a generic “basic brick” having one or several implementations called *submodels*, each of them specifying the *causality* attached to each of its port signals. In this paradigm, there is no fundamental difference between control signals and physical signals (i.e., signals held by *power variables* in corresponding bond graph models): it follows that native submodels can be used to implement control. However, while monolithic controllers (i.e., implemented as single submodels) can be made reasonably safe,<sup>1</sup> controllers built out of *smaller bricks* suffer from two weaknesses inherited from the submodel composition operation:

- the resulting flow of events is not synchronized and
- some control flow defects (resulting, for instance, in blocking models) cannot be detected at compile time.

As a consequence of the former, *cascades of events* are typical of models that deal with discontinuous—not necessarily piece-wise constant—signals. For instance, if a submodel’s job consists in converting its real piece-wise constant input to an integer, it will trigger a fresh event each time its output changes,<sup>2</sup> even if in this case—the input is piece-wise constant—we know that instants corresponding to output changes form a *subset* of those corresponding to

<sup>1</sup>We’ll come back to them in subsequent sections when talking about automatic code generation for controllers.

<sup>2</sup>This is necessary in order to notify a change to possible listener submodels.

input changes.<sup>3</sup> This has unfortunate consequences over resulting models: for instance, it is not possible to know, when a bunch of events fire, whether these events trace back to the same cause or not. Consider our real-to-integer converter example: since each jump in the output signal slope triggers a fresh event in disregard of the reason that made the jump necessary (actually another jump, so another event) we end up having to deal with *two simultaneous* events.<sup>4</sup> Even if consequences over performance are generally negligible, models have to figure out somehow that, given a bunch of events, some of them are “duplicates” of others to avoid treating each of them as independent events, yielding wrong results in some circumstances (see [2] for concrete examples of such wrong models). So to avoid practical synchronous issues, some form of collaboration between submodels—a *design pattern*—must be implemented by *library developers* and, moreover, *understood* and *correctly used* by end-users. Indeed, this collaboration scheme is unknown from the modeling tool which is then of no help to track down misuses of the submodels. This is arguably too much to require from both library developers and end-users, who should ideally focus on physics and control, rather than on low-level implementation details.

Also, it would be desirable to be able to statically (i.e., before execution) catch modeling errors resulting in *non-deterministic models* and *fragile models*.<sup>5</sup> Alas, the technological-and-phenomenological approach is again of no help here: in this paradigm, such modeling errors can only be detected—if they ever are—at runtime. As a consequence, development of models involving many discrete states can become cumbersome: it requires extensive testing to gain confidence in the correctness of the design, and whenever an error is detected, the location of the faulty submodels can be quite challenging. On the other hand, modeling with discrete states is precisely where the state-and-transition approach shines: when the state-and-transition graph of (part of) a model is explicit, it is possible to detect *at compilation time*, as we shall explain in subsequent sections, non-deterministic and

<sup>3</sup>In synchronous language terminology, we say that the *clock* corresponding to output changes is a *subclock* of the clock corresponding to input changes.

<sup>4</sup>This example will be further discussed in subsection 4.1.

<sup>5</sup>Fragile models are models whose correct execution relies on a property that escapes the automatic checker’s proof capabilities. Some of those models are actually correct, but some others are not: we prefer to reject *all* suspicious models, forcing users to disambiguate correct ones (disambiguation is often easy) rather than accepting wrong models that may be hard to debug.

fragile *patterns* that may lead to runtime errors.

Clearly, the technological-and-phenomenological approach, despite its versatility, reaches its limits when complex discrete state submodels such as those involved in Control applications come into play. This observation has motivated the extension of AMESim’s submodel description capabilities, which now feature a state-and-transition perspective.

## 2.2 Statecharts: an intuitive yet expressive graphical language

The general adoption of (variants of) statecharts in Control tools is due to their ability to concisely express complex finite state machines, making them reasonably understandable by humans.

Conciseness is mainly achieved thanks to the nice concept of *composite state*, which can be seen, at least in the original proposal by Harel [1], as a kind of “pseudo-abstraction” in the sense that this construct effectively allows some details of the equivalent, un-factored, flat machine to be abstracted away, but it is nevertheless necessary to reveal some contents to allow inner transitions. Statecharts also feature discrete state variables (updated in *actions*), which makes them suitable to describe even infinite state machines. Actually, Harel’s statecharts come with many appealing features, so they constitute a very good starting point for our targeted applications. However, since they are historically strongly rooted in the discrete control world, they lack the concept of continuous state variable. So, like many others did before us, we have extended statecharts to support hybrid modeling. We have designed this extension so that it remains simple and intuitive, yet powerful enough to handle many practical applications.

We will review in the next section some theoretical aspects of timed systems that have guided integration of statechart modeling capabilities in AMESim. We will then dive in important technical achievements such as validation and automatic code generation before presenting the final result from an end-user point of view.

## 3 A variant of the statechart language

The graphical language implemented in AMESim is very similar to the original statechart language: a statechart is essentially a set of *states* represented with rectangular boxes (labelled `count` and `reset` in the

example of Figure 1) and a set of possible *state transitions* represented with labelled arrows.

A transition can be associated with a *trigger*, a *guard* and *actions*. A transition is taken when an event corresponding to its trigger occurs if its guard evaluates to true. In that case, the actions—which are *state variable* assignments—are executed. For instance, the transition from `count` to `reset` in Figure 1 labelled

```
e [counter == 10] /
  counter := 0;
  reset_count := reset_count + 1
```

means that when an event associated with `e` occurs while `count` is active and the state variable `counter` is equal 10, a transition from `count` to `reset` is taken, `counter` is set to 0 and `reset_count` is incremented.

Triggers, or event generators, are defined at statechart creation time by boolean expressions that create events on their rising edges.

In addition to these general features, a statechart can be augmented with inputs from and outputs to AMESim. An output can either refer to a state variable, i.e. a discrete variable, or to a continuous signal controlled by state activations.

## 4 Theoretical aspects of statecharts integration in AMESim

When Modelica-based tools *simulate* the dynamic behavior of a system, they actually try to find a reasonable approximation of the solution of a system of equations that is supposed to capture the behavior of interest. This system of equations generalizes the so-called state space representation: it roughly consists in inputs, outputs and internal variables constrained by sets of equations whose (possibly dynamic) activation determine the trajectories of the model. This form constitutes the actual *denotation* of the corresponding *desugared program*<sup>6</sup> from which tools need to deduce the desired approximation. Now, given a desugared program, how is this approximation actually obtained? Of course, at some point, it depends on design choices made in the simulation tool.<sup>7</sup> But, given a common model description language like Modelica, any implementation is supposed to follow the same *operational semantics* which state *how to compute the solution—not an approximation—of any well-behaved program*:

<sup>6</sup>“Flat program” in Modelica.

<sup>7</sup>A tool may favor one or several classes of problems (e.g., marginally stable problems, discrete problems, etc.). This contributes to the tool’s added value.

this is somewhat the “reference implementation”—although a “virtual” one<sup>8</sup>—of a correct interpreter of the modeling language. Unfortunately, in practice, defining sound operational semantics for a physical system modeling language is such a huge work that, for most languages available today (including Modelica), only *informal semantics* (i.e., given in written human language) are available. Consequently, many inconsistencies simply cannot be spotted, because, contrary to formal descriptions, informal descriptions do not easily allow *reasoning* about the semantic model itself. This partly explains why, as pointed out in, e.g., [3], physical simulation tools experience difficulties in correctly handling some hybrid problems.

In AMESim, we had to take this fact into consideration when designing the new state-and-transition mode (which interacts with continuous behavior) so that we practically avoid most issues encountered in an unchecked implementation.

In the following subsections, the issues involved with coupling statecharts with AMESim models and the design choices made to circumvent them are discussed.

#### 4.1 Synchrony vs. Simultaneity issues

Statecharts describe systems that *react* to the environment, i.e., they respond to external *triggers* by executing state transitions according to their internal state and their inputs. In other words, a statechart merely describes a function that computes a new internal state from a previous state and a set of inputs while the environment is responsible for providing the inputs and for deciding *when* this function should be used to compute a new state.

A first source of non-determinism originates from the fact that the execution of a statechart can sometime be triggered by several different event sources *at the same time*. The issue is then to tell if the corresponding events are actually one and the same, i.e. they are dependent, or *synchronous*, events, or if they are unrelated, i.e., *asynchronous*, and actually occur in sequence but just seem to be simultaneous due to numerical approximations.

As an example, let us consider the simple statechart of Figure 2.

It simply says that

- if the event  $e1$  occurs while in state  $s0$ , the statechart transitions from  $s0$  to  $s1$ ,

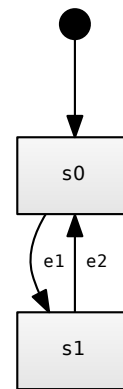


Figure 2: A simple statechart

- if the event  $e2$  occurs while in state  $s1$ , the statechart transitions from  $s1$  to  $s0$  and
- the statechart starts with  $s0$  being active.

If the statechart is executed because only one of  $e1$  or  $e2$  has occurred, the computation of the new state is quite straightforward. But what should happen if, for example,  $s0$  is active and both  $e1$  and  $e2$  are sensed simultaneously? Different interpretations are possible.

A first interpretation might be to assume that events that occur simultaneously are *exactly* the same, i.e. that they originate from the same *primary* external source and should consequently only be taken into account once. In the example this means that the new active state should be  $s1$ .

Another interpretation could be that  $e1$  and  $e2$  are *independent* and that the fact that they appear simultaneous is just a numerical artefact. In that case, the statechart would need to be executed twice according to the sequence in which  $e1$  and  $e2$  have occurred. If  $e1$  happens first,  $s1$  would become the active state after the first execution and  $s0$  would again be after the second one. It should be noted that in this interpretation it is not enough to know that the events are independent: the order in which events occur needs to be determined as well.

Both interpretations can be perfectly valid depending on the external context. Let us reuse examples similar to those presented in [2] to illustrate this fact.

Let us assume that the events  $e1$  and  $e2$  occur when some external signals  $i1$  and  $i2$  respectively become larger or equal to zero.

Figure 3 shows an AMESim model where the same signal source is connected to both  $i1$  and  $i2$ . The source defines a piece-wise constant signal that is  $-1$  when  $t < 1$  and  $1$  when  $t \geq 1$ . As a result, events  $e1$  and  $e2$  happen simultaneously when  $t = 1$ . In that

<sup>8</sup>This guarantees implementation independence.



case, it would make sense to consider that  $e_1$  and  $e_2$  refer to the same events and to adopt the corresponding interpretation in the statechart.

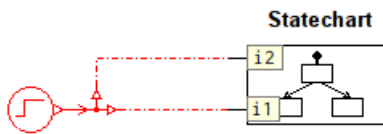


Figure 3: Dependent event sources

Let's now consider the model shown in Figure 4. The same statechart model is fed the outputs of position sensors attached to two identical mass with viscous friction models. If the velocities and positions of the masses are initialized to the same values, both positions will become non-negative simultaneously (if they ever do). However, it does not make sense to consider that these events are related as the models that generate them have nothing to do with each other. It seems much more natural to take them into account one after the other as if they had been sensed in sequence. The order of this sequence is not that important here as there is absolutely no reason to favor one model over the other. Actually, in real life, two seemingly identical systems submitted to the same inputs will always behave slightly differently at some scale because of uncontrolled parameters.

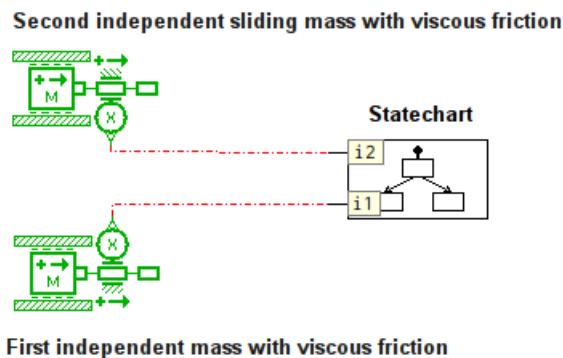


Figure 4: Independent event sources

A third way of using the statechart of Figure 2 is presented in Figure 5. This model illustrates the idea expressed in Subsection 2.1 about cascading events: a piece-wise constant signal is fed to  $i_1$  while its integer part is fed to  $i_2$ . Both inputs cross zero at the same time, generating simultaneous events. One could consider that they are the same events, as in the model in Figure 3, but one could also consider that  $e_2$  is a *consequence* of  $e_1$  and should then be handled *after*  $e_1$ .

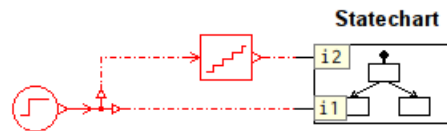


Figure 5: Independent event sources where order matters

The examples presented above show that very different meanings can be given to simultaneous events. Unfortunately, a continuous-time modeling environment is unable to give any insight about which is the expected one, the information being simply unavailable. This shortcoming is particularly critical when executing a statechart as making wrong decisions in a discrete model can radically alter the course of a simulation compared to continuous-time detailed physical models where energy conservation principles make models more robust with regard to non-determinism.

## 4.2 Observability of state transitions

In AMESim, just like in Modelica, discrete states can only be assigned *once* when a continuous-time event occurs.<sup>9</sup> This makes up another obstacle in the way of coupling a statechart with a continuous-time model. Indeed, a statechart may need to execute several transitions *without increasing the elapsed time in the model*, thus updating its state multiple times as a response to a unique continuous-time event.

Let us consider for example the statechart of Figure 6 featuring an input  $i$ , a discrete real output  $o$  and an event generator  $e$ . The statechart starts in state  $s_0$ . When  $e$  generates an event, state  $s_1$  is entered and  $o$  is set to 1.0. Besides, if the guard  $i > 0.0$  is satisfied at that time,  $s_2$  directly becomes the new state and  $o$  is set to 2.0, *without awaiting a new event*.

This clearly contradicts the assumptions stated above. In Modelica, for instance, something like the partial Modelica code in Listing 1 would need to be written, which is invalid as the second when clause introduces an algebraic loop involving  $s_1$  and because  $s_1$  and  $o$  are potentially constrained by two equations at once when  $e$  occurs.

Listing 1: Invalid Modelica code for the statechart of

<sup>9</sup>The values of certain states can actually be changed by a Modelica simulator as part of a solving process, but these intermediate values should never be relied upon in a model.

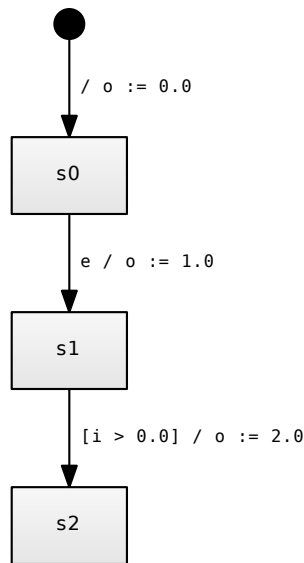


Figure 6: A statechart involving a potential double assignment

Figure 6

```

when initial() then
  s0 = true;
  s1 = false;
  s2 = false;
  o = 0.0;
elsewhen e then
  if pre(s0) then
    s0 = false;
    s1 = true;
    o = 1.0;
  end if;
end when;

// when state s1 is entered...
when s1 then
  if i > 0.0 then
    s1 = false;
    s2 = true;
    o = 2.0;
  end if;
end when;
  
```

One could, of course, think of working around these obstacles by “inlining” the intermediate transition, i.e. by rewriting the statechart with a direct transition from 0 to s2 as done in Listing 2.

Listing 2: Modelica code for the statechart of Figure 6 using “inlined” transitions

```

when initial() then
  s0 = true;
  s1 = false;
  s2 = false;
  o = 0.0;
elsewhen e then
  if pre(s0) then
    if i > 0.0 then
      s0 = false;
      s2 = true;
    end if;
  end if;
end when;
  
```

```

o = 2.0;
else
  s0 = false;
  s1 = true;
  o = 1.0;
end if;
end if;
end when;
  
```

This is valid Modelica, but the behavior is not exactly the expected one. Indeed, from an external point of view, *o* jumps from 0.0 to 2.0 directly without ever taking the 1.0 value. This may seem harmless for one used to physical continuous-time modeling, but what if this output was fed to another discrete part, e.g. another statechart, that relies on it to significantly alter its behavior? Figure 7 shows an AMESim model that depends on the intermediate value being taken. Event *e* is fired when the input becomes positive which at the same time satisfies the guard  $i > 0.0$ . The output is connected to a discrete subsystem that increments a counter when it gets exactly equal to 1.0. This means of course that the counter will only be incremented if the intermediate value is properly observed.

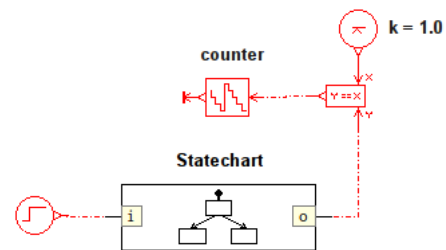


Figure 7: A model exposing observability issues

It should be noted that using Modelica algorithms to rewrite the model as shown in Listing 3 exposes the same issue as *o* is only equal to 1.0 during an intermediate step inside the algorithm.

Listing 3: Modelica code for the statechart of Figure 6 using an algorithm

```

algorithm
when initial() then
  s0 := true;
  s1 := false;
  s2 := false;
  o := 0.0;
elsewhen e then
  if pre(s0) then
    s0 := false;
    s1 := true;
    o := 1.0;
  end if;
end when;

// when state s1 is entered...
  
```

```

when s1 then
  if i > 0.0 then
    s1 := false;
    s2 := true;
    o := 2.0;
  end if;
end when;

```

### 4.3 Design choices

When designing the statechart extension to AMESim, the main focus was placed on the robustness, reliability and usability of the solution. That is why special care was taken to avoid as much as possible the issues presented above. This necessarily lead to additional rules in our variant of the statechart language that will be justified in this section.

First, it was decided to avoid the simultaneity issues described in Subsection 4.1 altogether by making sure that the execution of a statechart is always independent of the interpretation given to simultaneous events. This means that if any two events occur at the same time, assuming that they are synchronous or asynchronous should not change the upcoming computation. In the language described in this paper, events triggering a statechart simultaneously are simply forbidden.<sup>10</sup>

This rule is partially enforced by the statechart environment of AMESim by statically checking that a transition cannot generate an event that would conflict with the one that triggered the transition in the first place. What is more, simultaneous external events are detected at runtime and result in aborting the simulation. This guarantees the deterministic execution of a statechart provided that no dependent event sources are created by directly connecting—without inserting a continuous or discrete state variable as a buffer—an output to an input.<sup>11</sup>

One may argue that another solution might have been to stick to one interpretation anytime events occur simultaneously. But what if the only interpretation that makes sense is precisely the other one like in the model of Figure 4, where assuming that the events are dependent is clearly not expected? That is why signaling ambiguous situations was favored over making arbitrary choices behind the scene. As a side note, the “independent events” interpretation brings its lot of additional questions. How can the ordering of events be determined? How can several events be processed without

<sup>10</sup>This rule may be made less restrictive in the future if it turns out that it significantly enlarges the range of valid models.

<sup>11</sup>Avoiding this is the responsibility of the environment and cannot be enforced locally in a statechart.

increasing the continuous time and without encountering the issues of Subsection 4.2?<sup>12</sup>

Similarly, to avoid the observability issues presented in Subsection 4.2, the decision was made to enforce that the *effect* of every taken transition can always be observed from outside a statechart. This means that an *output* cannot be set more than once during one execution of a statechart. This property can be checked statically assuming that the first rule about simultaneous events is enforced.

## 5 Practical validation of statecharts

To ensure the safe execution of a statechart definition, the fulfilment of the aforementioned constraints has to be statically checked before generating code. Beside trivial checks such as making sure that the state machine contains a unique initial state or that at least one event generator is present (to ensure time progression), a few non-trivial checks have also been implemented. These checks are presented hereafter.

### 5.1 Transition expression checking

Writing correct transition expressions is error-prone: as in any other textual language, one can easily make syntax and semantic errors such as referring to a non-existent variable or event generator, or combine incompatible expressions.

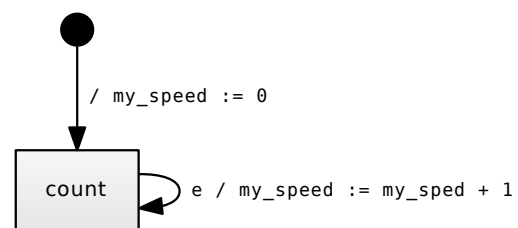


Figure 8: A statechart with a typo in an identifier

In order to avoid runtime errors due to unknown identifiers, each input, output and event must be declared before use. All local variables have to be initialized, and thus implicitly declared, on the initial transition of a statechart. These rules imply that all variables are known at compile time and allow rejecting the statechart of Figure 8 where `my_sped` is not a valid variable name.

<sup>12</sup>For instance, a state transition may change an output that in turn invalidates an event that has already been placed in the processing queue.

Avoiding element misuse requires a more complex solution: a static type system. This type system was designed to avoid the need for user provided type annotations, which would make writing transition expressions cumbersome. The Hindley/Damas/Milner type inference algorithm [4, 5] is used to type check all transitions without type annotations.

The basic idea of type inference is to traverse all expressions of a program to gather various typing constraints and correlations and then to resolve all those constraints in a second pass, thus attributing their final types to variables.

Let us consider for example the following expression

$$e / v := 2 + v; x := \sin(v) + x$$

and the primitive type definitions below.

$$\begin{array}{l|l} \sin & \text{Real} \rightarrow \text{Real} \\ 2 & \forall \alpha \in \{\text{Integer}, \text{Real}\}, \alpha \\ (+) & \forall \alpha \in \{\text{Integer}, \text{Real}\}, \alpha \rightarrow \alpha \rightarrow \alpha \end{array}$$

The first step to perform type inference on the example expression is to attribute free type variables to the variables in use, i.e.  $v$  will get type  $\beta$ ,  $x$  type  $\delta$  and  $e$  type  $\gamma$ .

As  $e$  is used in the trigger section of the expression, the typer can deduce the constraint  $\gamma = \text{Event}$ , which gives the final type for  $e$ : Event.

In the first action, the type of the  $(+)$  operator implies that both its operands must have the same type, which can be either Integer or Real. This means that the type of variable  $v$  must satisfy the following constraint:

$$\beta \in \{\text{Integer}, \text{Real}\}.$$

On the other hand, the type constraint for  $2$  matches exactly the one for the operands of  $(+)$  and can then be omitted in the next equivalences as it brings no additional information.

Similarly, the use of the  $(+)$  operator in the second action yields the following constraint:

$$\delta \in \{\text{Integer}, \text{Real}\}.$$

The presence of the  $\sin$  function generates tighter constraints, as it bounds the types of the input and output variables, yielding:

$$\begin{array}{l} \beta = \text{Real}, \\ \delta = \text{Real}. \end{array}$$

Gathering all the inequalities together gives the following final typing equation system:

$$\begin{array}{l} \beta = \text{Real}, \beta \in \{\text{Integer}, \text{Real}\}, \\ \delta = \text{Real}, \delta \in \{\text{Integer}, \text{Real}\}. \end{array}$$

It can then be simplified to deduce that  $v$  has type Real and that  $x$  also has type Real. An impossibility to simplify type equations, like obtaining  $\text{Integer} = \text{Real}$  would have meant a type error which should be reported to the user.

### 5.2 Activation chain analysis

Transitions that have no trigger section do not necessarily stop the execution of a statechart; they are taken as far as their guards allow, without waiting for another event.

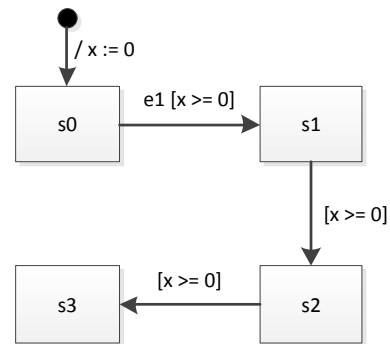


Figure 9: An activation chain

The execution of the statechart of Figure 9 exhibits such a behavior: when the event  $e1$  is raised while in  $s0$ , the state machine will take the transition to  $s1$  as  $x$  is equal to 0, then take the transition to  $s2$ , as the transition has no trigger, and finally to  $s3$ . To an external observer, the visible state activation will go from  $s0$  to  $s3$  directly. A path made of transitions that can be taken globally during an execution of a statechart is called an activation chain.

This behavior can result in infinite looping if not handled carefully, like in Figure 10. The semantics of our statechart language imply that the execution will continue indefinitely between  $s0$  and  $s1$  (highlighted in red), without ever resuming continuous-time simulation.

To ensure that a statechart will never stall a simulation, activation chain cycling is forbidden, hence forcing users to break cycles with triggers. For example, fixing the statechart of Figure 10 requires the addition of a trigger on the transition between states  $s1$  and  $s0$ .

The possibility to take several transitions in a single execution leads to potential duplicated assignments of variables, as in Figure 6 where variable  $o$  is assigned

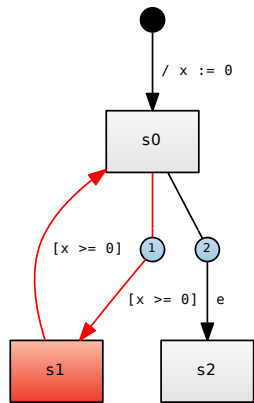


Figure 10: A cyclic activation chain

twice, once on the transition from  $s_0$  to  $s_1$  and once on the transition from  $s_1$  to  $s_2$ .

As explained in Subsection 4.2 about transition observability issues, this behavior is undesired and the implementation prevents it by analyzing the assignments along every activation chain and rejecting the activation chains that assign a variable more than once.

### 5.2.1 Practical chains analysis

The algorithm used to check the invariants mentioned above is mainly a depth-first search coupled with memoization. Each state is visited to compute its activation chains and forward assigned variables while the set of visited states in the current activation chain and the set of already assigned variables are maintained. Any intersection with the visited values and the ones already stored results in an error.

Another possibility would be to use a data-flow framework and express activation and assignment as liveness information to be propagated by the framework.

## 6 Code generation strategy

In this section, an overview of a code generation strategy leveraging the existing AMESim Modelica tool chain is presented.

### 6.1 Describing statecharts in Modelica

Modelica fits well to our purpose as a statechart that passes the validation stages discussed in the previous section can be described by a Modelica model that is:

- *valid*, i.e., is guaranteed to compile without error, thanks to the syntax and type check phases, and

- *sound* with respect to mixed discrete/continuous semantics as all ambiguous models are filtered out by the restrictions regarding simultaneous events.

Listing 4 shows how Modelica code can be generated to describe the statechart of Figure 1.

Listing 4: Code generated for the statechart of Figure 1

```
e = pulse > 0.0;
when initial() then
  reset_count = 0;
  counter = 0;
  st2 = true;
  st1 = false;
elsewhen e then
  if pre(st2) then
    if pre(counter) == 10 then
      counter = 0;
      reset_count = pre(reset_count) + 1;
      st2 = false;
      st1 = true;
    else
      counter = pre(counter) + 1;
      reset_count = pre(reset_count);
      st2 = true;
      st1 = false;
    end if;
  elseif pre(st1) then
    counter = pre(counter) + 1;
    reset_count = pre(reset_count);
    st2 = true;
    st1 = false;
  else
    counter = pre(counter);
    reset_count = pre(reset_count);
    st2 = pre(st2);
    st1 = pre(st1);
  end if;
end when;
```

Each event generator is associated with an `elsewhen` clause with the logic to handle this specific event as the body of the clause. This logic is encoded as cascading `if` statements representing the various transitions, the transition priorities being used to order the conditions. The equations describe the accumulated content of the action part of every transition and of the state activation updates.

Activation chains are generated recursively, for each level of the chain, the final body standing for the concatenation of all the actions. For instance, adding a transition from `reset` to `count` with the expression `[reset_count > 2]` and a priority below the existing one in the example counter statechart would create an activation chain resulting in the code shown in Listing 5.

Listing 5: Code generated for the statechart of Figure 1 augmented with an activation chain

```
...
```

```

if pre(st2) then
  if pre(counter) == 10 then
    if pre(reset_count) > 2 then
      counter = 0;
      reset_count = pre(reset_count) + 1;
      st2 = true;
      st1 = false;
    else
      counter = 0;
      reset_count = pre(reset_count) + 1;
      st2 = false;
      st1 = true;
    end if;
  else
    counter = pre(counter) + 1;
    reset_count = pre(reset_count);
    st2 = true;
    st1 = false;
  end if;
elseif pre(st1) then
  ...

```

## 7 Graphical user interface aspects

AMESim provides an editor to let users easily create statecharts and statically validate them, highlighting erroneous parts in red as seen in Figure 11.

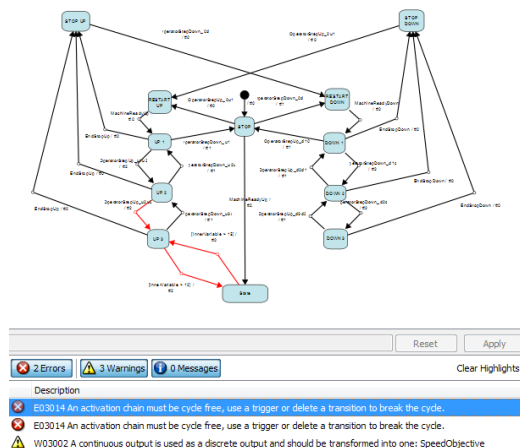


Figure 11: A statechart with a structural error highlighted in red

The editor also serves as a debugger as it is able to replay the behavior of a statechart during simulation, highlighting the active states and showing the values taken by all variables. Additional post-processing features are available, such as the possibility to easily jump between state changes. Timing diagrams representing state activations can be obtained using the regular AMESim data plotting facilities by displaying the activation variable associated with every state.

## 8 Conclusion and perspectives

In this paper, a few challenging issues associated with modeling and simulating hybrid models in an asynchronous environment are discussed.

A practical solution implemented in AMESim to avoid those issues and to offer a reliable and usable user interface is presented. This solution is built on top of the AMESim Modelica tool chain and demonstrates how a specific language can be implemented in terms of a more general language. The condition is, of course, that any model expressed in the specific language can somehow also be expressed without loss of meaning in the base language, which demands special care. However, the advantages of this approach are appealing as it is then possible to combine the convenience associated with a user friendly dedicated language with the power of a more general underlying language allowing to connect models expressed using different paradigms together. A core language can thus be extended to new applicative domains without being altered, retaining its generality.

Future work involves extending the supported subset of the statechart language, for example to allow parallel states. Developing practical and scalable means of building statecharts by composing smaller ones is another interesting perspective.

## References

- [1] David Harel. Statecharts: A visual formalism for complex systems, 1987.
- [2] Sébastien Furic. Enforcing reliability of discrete-time models in modelica. In *Proceedings of the 8th International Modelica Conference*, 2011.
- [3] Albert Benveniste, Timothy Bourke, Benoît Cailaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78(3):877 – 910, 2012.
- [4] R. Hindley. The Principal Type-Scheme of an Object in Combinatory Logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.
- [5] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212, New York, NY, USA, 1982. ACM.

# Integration of OpenModelica in Ptolemy II

Mana Mirzaei   Lena Buffoni   Peter Fritzson  
Department of Computer and Information Science (IDA),  
Linköping University, Division SE-581 83, Linköping, Sweden

## Abstract

In this paper we present the work done to integrate OpenModelica into the Ptolemy II framework for modeling large-scale concurrent systems. To this end a dedicated computational model for OpenModelica has been defined in Ptolemy II, and support for tool-interaction has been implemented. This implementation will allow to simulate existing Modelica models by the OpenModelica compiler in a heterogeneous context together with models from other computational domains.

*Modelica, Ptolemy II, hierarchical system modeling, concurrent systems*

## 1 Introduction

Distributed, concurrent systems are becoming increasingly common. However, they are complex to develop. Therefore a large number of computational models and tools for modeling and developing such systems has emerged. The Ptolemy project aims to support such heterogeneous modeling in Ptolemy II[1], an open-source software framework for modeling, simulation and design of large concurrent real-time systems. This framework is a system-level design environment that provides the possibility of combining several variants of models of computation (MoCs) in one hierarchical heterogeneous model. Ptolemy II also supports an actor-oriented view of a system where the basic building blocks of a system are concurrent components called actors which communicate through messages sent via interconnected ports.

Ptolemy II currently supports numerous concurrent programming models, such as process networks, discrete event or continuous time models and interfaces with other simulations tools such as Matlab [2]. In this paper we present the integration of Modelica models in Ptolemy II through an integration with the OpenModelica tool.

Modelica[3] is a non-proprietary, object-oriented,

equation based language aimed at modeling complex multi-domain physical systems. Moreover, this language is supported by a number of free and commercial tools, in particular by OpenModelica [4], an open source compiler and tool suite, complete with a text and graphical modeling editor (OMEdit) for modeling and simulation of cyber-physical systems.

The Modelica Standard Library contains a large number of models and functions from multiple domains. Integrating Modelica in Ptolemy would enable the use of these models in concurrent system modeling, as well as models from many other existing libraries.

In this paper we present the integration process of OpenModelica into the Ptolemy II framework, and illustrate use of Modelica models from within Ptolemy II on a simple example.

The paper is structured as follows, Section 2 presents the basic blocks of the integration between OpenModelica and Ptolemy, Section 3 discusses the simulation mechanism, Section 4 presents the performances of the proposed implementation, Section 5 highlights some related works and finally Section 6 briefly sums up the contents of the paper.

## 2 Integration Architecture

Ptolemy II is composed by a number of different domains and supports an actor-oriented modeling paradigm. Actors are the main building blocks of the system. They are concurrent components that communicate through interfaces called ports. Relations define the interconnection between these ports, and the communication structure between actors. Viewing a system as a structure of actors emphasizes its causal structure and its concurrent activities, along with their communication and data dependencies. A consequence of an actor-oriented view of a system is the decoupling of the transmission of the data from the transfer of control [5].

## 2.1 OpenModelica director

Ptolemy II is modular and relies on a well-organized package structure where the core packages support the data model, or abstract syntax, of Ptolemy II design. These packages also support the abstract semantics that allows domains to interoperate with maximum information abstraction. The user interface packages provide support for our XML file format, called MoML, together with a visual interface for constructing models graphically. The library packages provide actor libraries that are domain polymorphic, meaning that they can operate in a variety of domains. Additionally, the domain packages provide domains, each of which implements a model of computation. Some of which provide their own, domain-specific actor libraries.

A model of computation (MoC) is defined as a set of rules that governs the interactions between components and determines the semantics of a model. Moreover, these rules determine when actors perform internal computation and update their internal state. The semantics of the computation model are implemented through the concept of *Director*.

In order to integrate OpenModelica into Ptolemy II, it is necessary to create a dedicated computation domain with the corresponding director. Since Modelica is a language designed for continuous and discrete event modeling modeling of physical systems and variables described using DAEs, the continuous-time domain in Ptolemy II which models physical processes and supports mixtures of discrete and continuous behaviors is considered the most suitable for the Modelica language. Therefore the OpenModelica domain extends the continuous time domain already present in Ptolemy. Figure 1 shows the OpenModelica director with its parameters, such as the number of iterations or the solver to be used .

The integration work described in this paper involves three sub-packages of the domain package: `kernel`, `demo` and `lib`. The kernel package provides the software architecture for the Ptolemy II data model, or abstract syntax. This abstract syntax has the structure of clustered graphs. The classes in this package support entities with ports, and relations that connect the ports.

## 2.2 OpenModelica actor

Actors, the basic building blocks of a system, are the executable entities used to build the models. The OpenModelica actor reads one or more inputs from

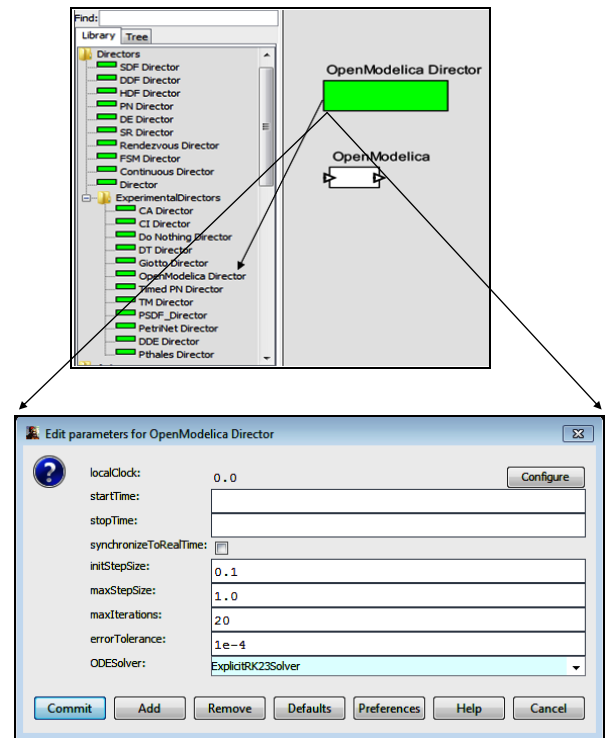


Figure 1: OpenModelicaDirector is provided on the left menu under the Directors → ExperimentalDirectors.

other actors and simulates the Modelica model. This simulation can be done in either batch or interactive processing mode. The results of the simulation can then be passed to Ptolemy II actors, to be displayed or modified.

An execution in Ptolemy II is divided into the following phases: setup, iterate, and wrap-up. The setup phase is divided into two phases, preinitialize and initialize. The preinitialize sub-phase usually handles structural information, such as constructing dynamically created actors, determining the width of ports, and creating receivers. The initialize phase initializes parameters, resets local states, and produces initial tokens. The preinitialization and initialization of an actor are performed exactly once during the actor’s life cycle.

To organize the interactions among actors, an iteration is divided into **prefire**, **fire**, and **postfire**.

1. Prefire checks if the preconditions are fulfilled for the actor to execute, such as the presence of sufficient inputs to complete the iteration.
2. Most of the actions are taken place in the fire phase, which involves reading the inputs, processing data, and producing outputs. Some MoCs



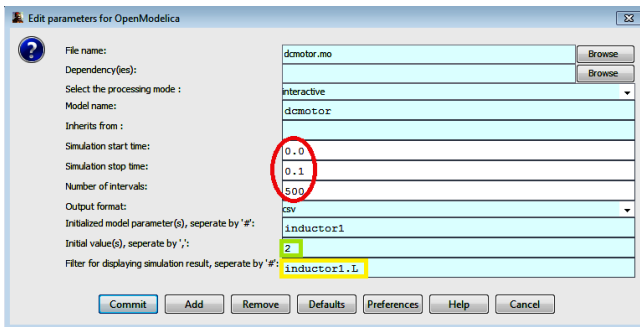


Figure 2: We can see that the model to be loaded is contained in the file *dcmotor.mo*. We can also set the simulation parameters (in red), the parameters of the model (green), such as for instance the resistance of the resistor and apply a filter to select the variables that should be displayed (in yellow).

```

model dcmotor
  Modelica.Electrical.Analog.Basic.Resistor resistor1(R = 10);
  //Observe the difference between MSL 2.2 and 3.1 regarding the default
  Modelica.Electrical.Analog.Basic.Inductor inductor1(L = 0.2);
  Modelica.Electrical.Analog.Basic.Ground ground1;
  Modelica.Mechanics.Rotational.Components.Inertia load(J = 1);
  // Modelica.Mechanics.Rotational.Inertia load(J = 1); // Mode
  Modelica.Electrical.Analog.Basic.EMF emf1;
  Modelica.Blocks.Sources.Step step1;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1;

```

Figure 3: The parameters for the Modelica model can be set through the graphical interface in Vergil.

like synchronous reactive models and CT differential equations support fixed-point iteration which enables the computation of the fixed point of actor outputs while keeping the state of each actor constant.

3. Invocation of `fire()` several times prior to the invocation of `postfire()` results to updating the state of an actor at the time of reaching the fixed point. At the final stage, `wrapup()` is invoked to release resources that were used during execution.

The domain specific OpenModelica actor is defined in the `lib` sub-package. It is an atomic actor which is visible in Vergil [6], the graphical editor for the Ptolemy II framework.

In this paper, we illustrate the use of the OpenModelica actor to simulate a simple DCMotor example. When looking inside the actor, all the simulation parameters are displayed, as illustrated in Figure 2. When this model is parametrized through the Vergil interface, this will initialize the corresponding Modelica model (Figure 3).

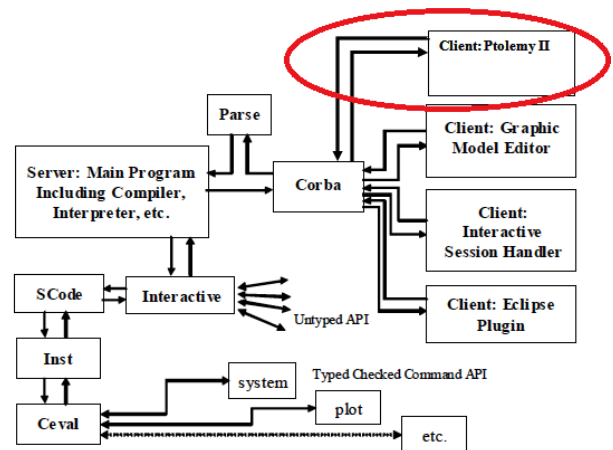


Figure 4: Client-Server interconnection structure of the compiler/interpreter main program and interactive tool interfaces after adding new Ptolemy II as a new client.

### 3 Simulation

To simulate Modelica models in Ptolemy II, it is necessary to invoke the OpenModelica Compiler (OMC). OMC provides a CORBA interface for remotely invoking the compiler from client applications. This interface is used to communicate with OMC from Ptolemy II (see Figure 4).

An OpenModelicaDirector (Figure 1), which extends the ContinuousDirector, is implemented in the Kernel package. The key function of this actor is `postfire()`, a method that will be invoked exactly once during an iteration, after all invocations of the `fire()` method in that iteration. However, in this integration, when `postfire()` is invoked in OpenModelicaDirector it returns false in order to stop/halt the invocation of `fire()` after firing once.

The communication between OpenModelica and Ptolemy follows the following pattern:

1. The OpenModelica actor starts the OpenModelica Compiler(OMC) server in the `initialize()` method, by invoking `startServer()`.
2. In the next step, value(s) of the Modelica parameter(s) are modified by Ptolemy IIs' actor(s) prior to the simulation of the Modelica model through OMC.
3. After simulation of the Modelica model, the retrieved result back from OMC server is plotted/displayed through Ptolemy IIs' actors.
4. Finally, OpenModelica actor sends the result to

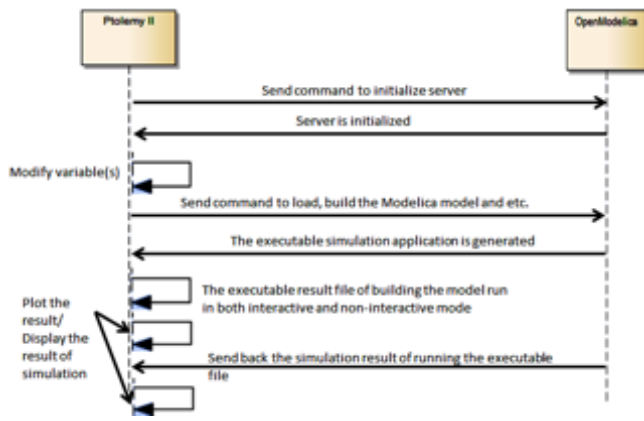


Figure 5: The interactions between Ptolemy II and OpenModelica can be summarized by the following diagram.

the output port which enables using the generated result by other Ptolemy IIs' actors.

All these actions are invoked in the `fire()` method. In the final step, the OMC server is halted through invoking `stopServer()` in the `wrapup()`. These steps are summed up in Figure 5.

The `OMCCCommand` class provides the implementation for all the functionalities required by the `initialize()`, `fire()` and `wrapup()` methods, except for modifying the component values after simulation. This functionality is provided by the `UntilSocket` class. Figure 6 presents the class diagram of the implementation, which contains around three thousand lines of code. `OMCCCommand` is the largest class, containing over a thousand of code lines. Since the Ptolemy II framework is written in Java, the `OpenModelica` extensions are also done in Java. Appendix A includes a code snippet that shows the implementation of the `fire()` method.

In order to integrate the `OpenModelica` actor with components for displaying the simulation results in ptolemy, a composite actor is constructed. Figure 7 shows the Composite actor that is used for plotting CSV format, the `OpenModelicaDirector` that controls the execution order of the `OpenModelica` actor and the `RunCompositeActor`, and `SDF` director which is known as the inside director of `RunCompositeActor` controls the execution of `CSVReader`, `RecordDisassembler` and `XYPlotter` whenever `RunCompositeActor` is executed.

In the current implementation both batch and interactive simulation are possible.

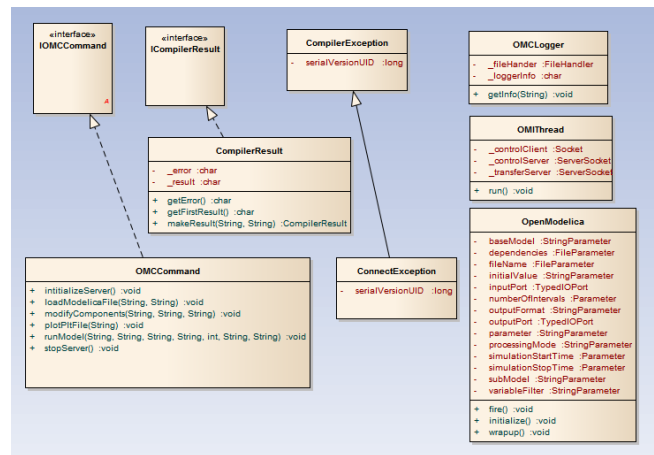


Figure 6: The class diagram for the current implementation.

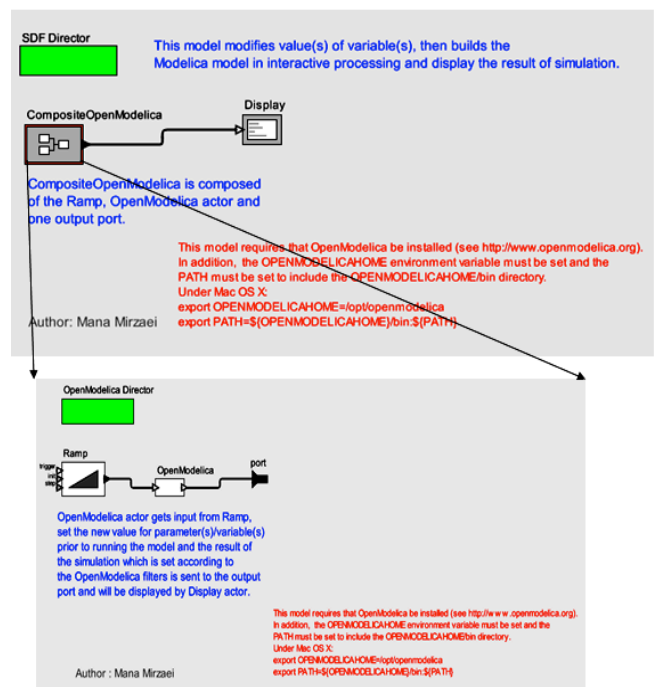


Figure 7: OpenModelicaXYPlotter model is composed of OpenModelica actor and RunCompositeActor.

### 3.1 Interactive Simulation

OpenModelica offers a user-interactive and time synchronous simulation known as OpenModelica Interactive (OMI). OMI is part of the simulation runtime core. The output of OMI is an executable simulation application, running the executable file in an interactive processing mode that enables users to govern the simulation runtime behavior.

The integration of OMI with Ptolemy is implemented through the following two modules, illustrated in Figure 8:

**Control module** is the interface between OMI and Ptolemy II which is implemented as a single thread to support parallel tasks and independent reactivity. The Control module is considered the major controlling and communication instance during the simulation initialization phase as well as for managing simulation properties throughout the simulation runtime. The Control module also reacts to the feedback from other internal OMI components and sends some messages back to Ptolemy II including error and status messages.

**Transfer module** gets simulation results from a result manager and sends them to Ptolemy II upon launching a simulation. Additionally, the module employs a filter mask allowing the user to select the variables whose result values are significant to Ptolemy II.

At the moment only the step by step simulation of the OpenModelica models is possible, however the next step in the development process is to implement cosimulation. Figure 9 illustrates the results of an interactive simulation, new value(s) for variable(s) of the Modelica model can be set in the *initial value(s)* parameter or the Ramp actor parameter which can be customized in the same way as the OpenModelica actor. The value of the *init* parameter of the Ramp actor overrides the value of *initial value(s)* parameter of OpenModelica actor.

The step time of the simulation in interactive processing is calculated by following formula:  $Simulation\ step\ time = (Simulation\ stop\ time - Simulation\ start\ time) / (Number\ of\ intervals)$ .

According to the above formula, the step time is 0.0002 [ (1.0 - 0.0)/500] in this example. The calculated step time results in the start of the simulation at 0.0002 and completion at 0.0998, as shown in red in Figure 9.

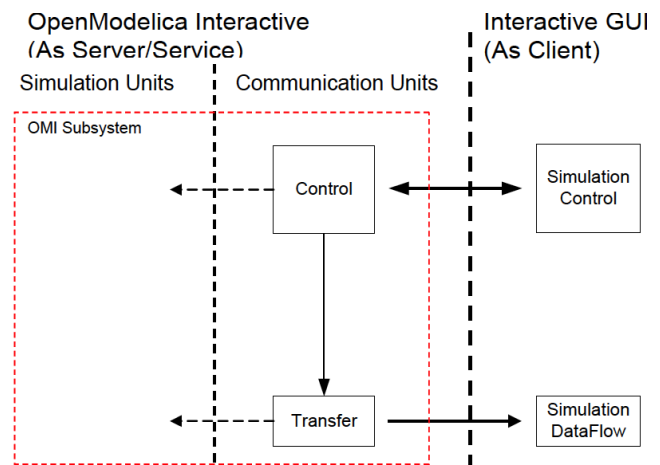


Figure 8: Interactive simulation mechanism.

```

.OpenModelica.Display
File Help
At time : 0.0002 resistor1.R_actual=2 load.w=2
At time : 0.0004 resistor1.R_actual=2 load.w=2
At time : 0.0006 resistor1.R_actual=2 load.w=2
At time : 0.0008 resistor1.R_actual=2 load.w=2
At time : 0.001 resistor1.R_actual=2 load.w=2
At time : 0.0012 resistor1.R_actual=2 load.w=2
At time : 0.0014 resistor1.R_actual=2 load.w=2
At time : 0.0016 resistor1.R_actual=2 load.w=2
At time : 0.0018 resistor1.R_actual=2 load.w=2
At time : 0.002 resistor1.R_actual=2 load.w=2
At time : 0.0022 resistor1.R_actual=2 load.w=2
At time : 0.0024 resistor1.R_actual=2 load.w=2
At time : 0.0984 resistor1.R_actual=2 load.w=2
At time : 0.0986 resistor1.R_actual=2 load.w=2
At time : 0.0988 resistor1.R_actual=2 load.w=2
At time : 0.0992 resistor1.R_actual=2 load.w=2
At time : 0.0994 resistor1.R_actual=2 load.w=2
At time : 0.0996 resistor1.R_actual=2 load.w=2
At time : 0.0998 resistor1.R_actual=2 load.w=2

```

Figure 9: Displaying part of simulation result of OpenModelicaInteractive.xml by the Display actor.

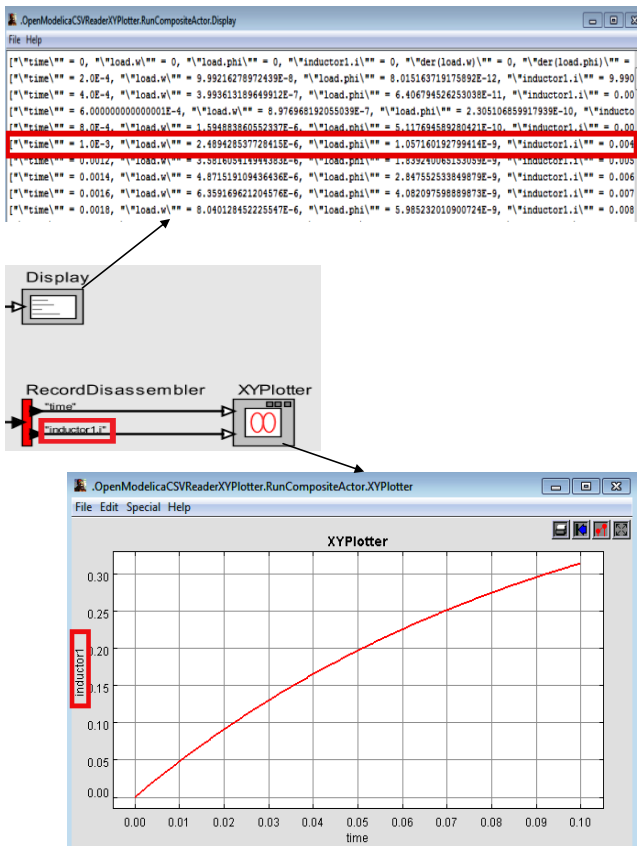


Figure 10: The generated output is displayed in XY-Plotter as well as in textual format by the correspond- ing actors.

### 3.2 Non-interactive simulation

There is also the possibility of running the executable simulation application which is generated by OMC in a non-interactive simulation runtime. In contrast to Interactive Simulation, this method does not offer any user-interactive simulation. Running the executable file in batch processing mode allows to generate the whole simulation result in CSV or PLT file formats (Figure 10) that can be displayed and used by other actors in Ptolemy II.

## 4 Performance

In this section we compare the simulation times of two test models in OpenModelica and when simulating them through Ptolemy, to estimate overhead. The following table compares the two:

Use Case	in Ptolemy II	in OpenModelica
OMXYPlotter	25021ms	30ms
OMPltPlotter	28673ms	12ms

As simulating the models in Ptolemy involves establishing a connection with the OpenModelica compiler and exchanging data, some overhead is to be expected. However it is the tradeoff for simulation in a heterogeneous environment.

## 5 Related Works

An integration between Dymola, one of the commercial tools for Modelica modeling and simulation, and Ptolemy II through a software environment known as BCVTB4 has been developed [7]. In this approach, Ptolemy II acts as the middleware for implementing BCVTB and actors in Ptolemy II are responsible for starting a server that uses the BSD5 socket utilized for exchanging data between the simulator and the actor as well as implementing the inter-process communication. This approach is implemented by adding BCVTB block to the Dymola library and it is necessary to include this block in the Modelica model in order to enable co-simulation.

FMI (Functional Mockup Interface) is an evolving standard for composing model components designed using distinct modeling tools that can also be used in the cosimulation of models [8].

## 6 Conclusion

In this article we have presented the integration of Modelica models in the Ptolemy II framework for modeling large-scale heterogeneous concurrent systems. This will allow the simulation of Modelica models through the use of the OpenModelica compiler within the Ptolemy II network.

Moreover the current architecture can serve as a base for further integration efforts, such as the bisimulation of Modelica models in conjunction with other formalisms.

## Appendix A

The following code snippet shows the implementation of the `initialize()` method:

```

/** Invoke the fire() of the super class. Then, Modelica library and model(s) are loaded.
 * Upon modifying the value of variable(s) and parameter(s) by input port or actors'
 * parameters,
 * the Modelica model is built in <i>non-interactive</i> or <i>interactive</i> mode.
 * <p>After building the model in an interactive mode, the simulation result
 * is calculated step by step according to the parameters of the OpenModelica actor.
 * The result is sent in the string format to the output port of the OpenModelica
 * actor to be
 * displayed by Display actor.</p>
 * @exception IllegalArgumentException If the evaluation of the expression
 * triggers it, or the evaluation yields a null result, or the evaluation
 * yields an incompatible type, or if there is no director.
 */
public void fire() throws IllegalArgumentException {
    super.fire();

    // Load Modelica library and model(s).
    try {
        _omcCommand.loadModelicaFile(fileName.getExpression(),
            subModel.getExpression());
        // If the model is inherited from a base model,
        // that base model should be loaded in advance to the derived model.
        // Otherwise, the derived one could not be built.
        if (!(dependencies.getExpression().isEmpty() && baseModel
            .getExpression().isEmpty()))
            _omcCommand.loadModelicaFile(dependencies.getExpression(),
                baseModel.getExpression());
    } catch (ConnectException e) {
        throw new IllegalArgumentException(
            "Unable to load Modelica file/library!" + e.getMessage());
    }

    // There is a value to be passed to the OpenModelica actor's port.
    if (input.getWidth() > 0) {
        // Get the token from input port of OpenModelica actor.
        IntToken inputPort = (IntToken) input.get(0);
        try {
            // Modify components of the Modelica model prior to running the model.
            if (!(parameter.getExpression().isEmpty() && initialValue
                .getExpression().isEmpty())) {
                if (!(baseModel.getExpression().isEmpty())) {
                    _omcCommand.modifyComponents(inputPort.toString(),
                        baseModel.getExpression(),
                        parameter.getExpression());
                } else {
                    _omcCommand.modifyComponents(inputPort.toString(),
                        subModel.getExpression(),
                        parameter.getExpression());
                }
            } else {
                _omcLogger
                    .getInfo("There is no component to modify prior to running the model!");
            }
        } catch (ConnectException e) {

```

```

        throw new IllegalArgumentException(
            "Unable to modify components' values!" + e.getMessage());
    }
    // There is no value to be passed to the OpenModelica actor's port and the new
    // value is set by
    // actors' parameters.
} else if (!(input.getWidth() > 0)) {
    if (!(parameter.getExpression().isEmpty() && initialValue
        .getExpression().isEmpty())) {
        try {
            if (baseModel.getExpression().isEmpty()) {
                _omcCommand.modifyComponents(
                    initialValue.getExpression(),
                    subModel.getExpression(),
                    parameter.getExpression());
            } else {
                _omcCommand.modifyComponents(
                    initialValue.getExpression(),
                    baseModel.getExpression(),
                    parameter.getExpression());
            }
        } catch (ConnectException e) {
            throw new IllegalArgumentException(
                "Unable to modify components' values of "
                + baseModel.getExpression() + " !"
                + e.getMessage());
        }
    } else {
        _omcLogger
        .getInfo("There is no components to modify prior to running the model!");
    }
}

// Build the Modelica model and run the executable result file.
// Plot the result file of the simulation that is generated in plt format.
try {
    if (!(dependencies.getExpression().isEmpty() && baseModel
        .getExpression().isEmpty())) {
        _omcCommand.runModel(dependencies.getExpression(),
            baseModel.getExpression(),
            simulationStartTime.getExpression(),
            simulationStopTime.getExpression(),
            Integer.parseInt(numberOfIntervals.getExpression()),
            outputFormat.getExpression(),
            processingMode.getExpression());

        if (outputFormat.getExpression().equalsIgnoreCase("plt")
            && processingMode.getExpression().equalsIgnoreCase(
                "non-interactive")) {
            _omcCommand.plotPltFile(baseModel.getExpression());
        }
    } else {
        _omcCommand.runModel(fileName.getExpression(),
            subModel.getExpression(),
            simulationStartTime.getExpression(),
            simulationStopTime.getExpression(),
            Integer.parseInt(numberOfIntervals.getExpression()),
            outputFormat.getExpression(),

```

```
        processingMode.getExpression());

    if (outputFormat.getExpression().equalsIgnoreCase("plt")
        && processingMode.getExpression().equalsIgnoreCase(
            "non-interactive")) {
        _omcCommand.plotPltFile(subModel.getExpression());
    }
}

// In case of building the model in an interactive mode, client and servers are
// created,
// IP and ports of the servers are set and streams for transferring information
// between
// client and servers are set up all in the constructor of the thread.
// Through starting the thread, the simulation result is sent from the server to
// the
// Ptolemy II in the string format.
if (processingMode.getExpression().equalsIgnoreCase("interactive")) {
    _omiThread = new OMIThread(variableFilter.getExpression(),
        simulationStopTime.getExpression(), output);
    // FIXME: This method explicitly invokes run() on an object. In general,
    // classes implement the Runnable
    // interface because they are going to have their
    // run() method invoked in a new thread, in which case Thread.start() is the
    // right method to call.
    _omiThread.run();
}
} catch (UnknownHostException e) {
    e.printStackTrace();
    throw new IllegalArgumentException("Host Exception: "
        + e.getMessage());
} catch (IOException e) {
    e.printStackTrace();
    throw new IllegalArgumentException("Socket Connection Error: "
        + e.getMessage());
} catch (ConnectException e) {
    e.printStackTrace();
    throw new IllegalArgumentException("ServerError: " + e.getMessage());
}
}
```

## References

- [1] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, Haiyang Zheng, Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II), April 1, 2008.
- [2] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, Haiyang Zheng, Heterogeneous Concurrent Modeling and Design in Java (Volume 2: Ptolemy II Software Architecture), April 1, 2008.
- [3] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pages, Wiley-IEEE Press, 2004.
- [4] OpenModelica Home Page, [www.openmodelica.org](http://www.openmodelica.org), Last Accessed Feb. 2013.
- [5] Johan Eker, Jörn W. Janneck, Edward A. Lee, Fellow, Ieee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, And Yuhong Xiong, Taming Heterogeneity-The Ptolemy Approach, Proceedings Of The IEEE, Vol. 91, No. 1, Jan. 2003.
- [6] UsingVergil, [http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIlatest/ptII/doc/design/usingVergil/using\\_Vergil.pdf](http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIlatest/ptII/doc/design/usingVergil/using_Vergil.pdf), Last Accessed June 2013.
- [7] Michael Wetter, Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed, Journal of Building Performance Simulation, Volume 4, Issue 3, 2011
- [8] David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis and Michael Wetter, Determinate Composition of FMUs for Co-Simulation, EECS Department University of California, Berkeley Technical Report No. UCB/EECS-2013-153 August 18, 2013



# On Extending JGrafchart with Support for FMI for Co-Simulation

Alfred Theorin    Charlotta Johnsson

Department of Automatic Control, Lund University, Lund, Sweden

## Abstract

Grafchart is a graphical programming language which extends Sequential Function Charts (SFC), the PLC standard languages for sequential, parallel, and general state-transition oriented automation applications. SFC is widely used and accepted for industrial automation. Grafchart adds higher level features to SFC such as hierarchical structuring, reusable procedures, and exception handling to make it convenient to implement and maintain large applications.

Functional Mock-up Interface (FMI) is a standard to combine dynamic system models for technical system developed in various tools. Tools can export models as Functional Mock-up Units (FMUs) which can be combined with other FMUs to compose the whole system.

In this paper adding *FMI for Co-Simulation* support to JGrafchart, a free implementation of the Grafchart language, is conceptually evaluated. It is discussed how JGrafchart fits into the *FMI for Co-Simulation* framework and potential ways to implement this are discussed. *Keywords: Grafchart; FMI; Co-Simulation; FMI for Co-Simulation; Modelica*

## 1 Introduction

Grafchart is a graphical programming language which extends Sequential Function Charts (SFC), one of the IEC 61131-3 [1] PLC standard languages for sequential, parallel, and general state-transition oriented applications. SFC is supported by most large industrial automation systems, for example 800xA by ABB, SIMANTIC S7 by Siemens, RSLogix 5000 by Rockwell Automation, DeltaV by Emerson, and CENTUM CS by Yokogawa. SFC is widely used and accepted for industrial automation, but is a low level programming language and thus implementing larger applications in SFC is inconvenient. Grafchart adds high level features such as hierarchical structuring, reusable procedures, and exception handling which makes it convenient to implement large applications that are

overviewable and maintainable [2].

Functional Mock-up Interface (FMI) is a recent standard [3] which aims at combining dynamic system models developed in various tools. Modelica [4], the state of the art language to express dynamic behavior of technical systems, promotes this standard and the number of tools supporting FMI is growing rapidly. A tool can export a model as a Functional Mock-up Unit (FMU) which can then be combined with other FMUs to compose the whole system. The FMI standard consists of two parts, namely *FMI for Model Exchange* and *FMI for Co-Simulation*. The difference is that for *FMI for Co-Simulation* a FMU also includes an individual solver to simulate its behavior.

In this paper adding *FMI for Co-Simulation* support to JGrafchart, a free implementation of the Grafchart language, is conceptually evaluated. In Section 2 *FMI for Co-Simulation* is described, in Section 3 the Grafchart and JGrafchart basics are covered, and Section 4 motivates the need to connect JGrafchart to *FMI for Co-Simulation* and discusses possible ways to implement this. Finally, future work is discussed in Section 5.

## 2 FMI for Co-Simulation

*FMI for Co-Simulation* is a standard which enables simulation of coupled technical systems with focus on time-dependent problems. It is designed for both standalone FMUs and FMUs which are FMI wrappers for simulation tools.

A co-simulation is executed from a given starting time to a stop time which is not necessarily pre-specified. There is an FMI *master* which coordinates the co-simulation and there are FMU *slaves*, each corresponding to one model or subsystem. Each *slave* has a pre-specified set of inputs and outputs which are known by the *master*. The *master* is responsible for initialization of the *slaves* and for handling the coupling between them by getting and setting their inputs and outputs.

The co-simulation is executed for one time interval at a time, known as a *communication step*, during which each *slave* executes independently. Between the *communication steps* are the *communication points* where the *master* communicates the inputs and outputs between the *slaves*. *Slaves* can specify their desired *communication step* size and the *communication step* size may also vary during the co-simulation provided that all *slaves* support this. A *communication step* may also fail. Then a new *communication step* of different size may be attempted if all *slaves* support redoing *communication steps*. It is the *master* which decides the *communication steps* and what to do when one fails.

The standard does not define an FMI *master* algorithm, and the level of sophistication is decided by the one who implements the *master*. What the standard does define is the API, a set of *slave* capabilities, and rules for how these may be used.

An FMU is described by an XML metadata file which primarily contains the inputs and outputs and co-simulation capabilities such as support for redoing *communication steps* and support for variable *communication step* size.

### 3 Grafchart

Grafchart has the same graphical syntax as SFC with steps and transitions, where steps represent the possible application states and transitions represent the change of application state. Associated with the steps are actions which specify what to do. Associated with each transition is a Boolean guard condition. It is a state machine related language which has been developed particularly with automation in mind and with focus on scalability.

A part of a running Grafchart application is shown in Figure 1. Here two steps are connected by a transition and there are two variables, namely *var* and *cond*. In the left part of the figure, the upper step has just been activated which involved executing its *S* action, thus setting *var* to 7. An active step is indicated by a black dot, known as a token. The upper step will remain active until the guard condition of the transition becomes true, that is, until *cond* gets the value 4. When the guard condition becomes true, shown in the right part of the figure, the upper step is deactivated and the lower step is activated which means that *var* is set to 12.

Steps also have additional properties, namely *x*, *t*, and *s*. *x* is true if the step is active and false if the step

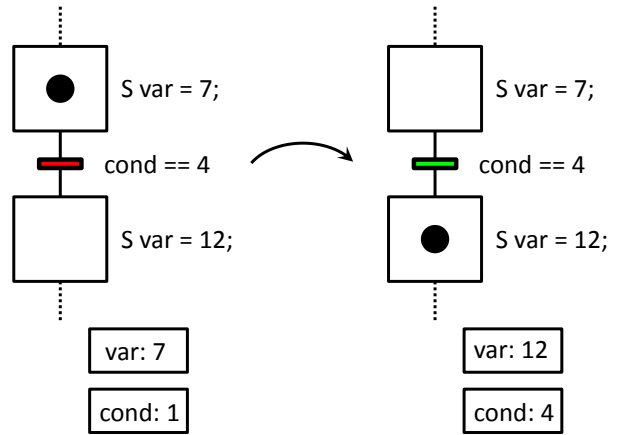


Figure 1: A piece of a running Grafchart application. The left part shows one application state and the right part shows a later application state.

is inactive. *t* is how many scan cycles the step has been active since the previous activation if the step is active. For inactive steps *t* is 0. *s* works the same as *t* but counts seconds instead of scan cycles.

Grafchart supports basic SFC functionality such as alternative and parallel paths, see Figure 2. At any time only one alternative path may contain active steps. On the other hand, parallel paths are executed in parallel and will contain active steps at the same time. To create alternative paths a step is connected to several transitions. To create parallel paths a Parallel Split is added to split the execution. A Parallel Join is used to merge the execution again when the parallel paths are completed.

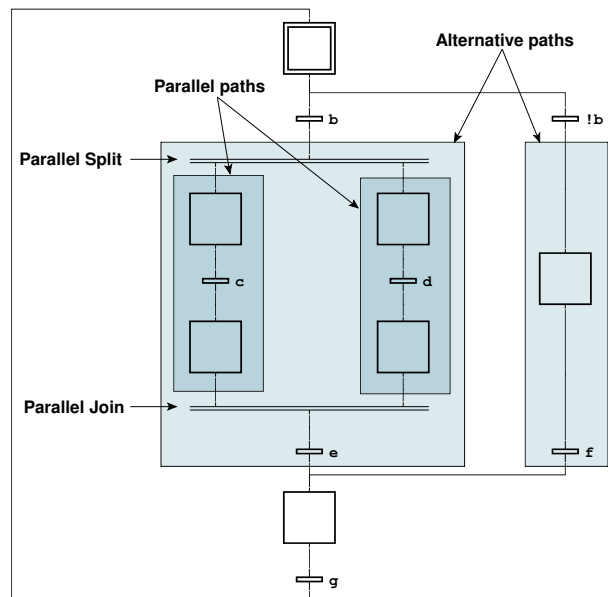


Figure 2: A Grafchart application showing how to express alternative and parallel paths.

In Figure 1 only the S action type is used. S actions are executed on step activation. SFC also supports several other action types (action qualifiers) which have other semantics. Grafchart supports fewer action types. However, these are more general and can, among other things, be used to implement the semantics of all action types of SFC. The main action types in Grafchart are S (executed on step activation), P (executed periodically while step is active), X (executed on step deactivation), and N (sets a boolean variable to true (false) on step activation (deactivation)).

Additional constructs such as hierarchical structuring, reusable procedures, and exception handling have been added in Grafchart which makes it convenient to implement large applications that are overviewable and maintainable [2].

With reusable components, code duplication is avoided. Reusable code can be put in a Grafchart Procedure which can then be called from any number of Procedure Steps and Process Steps, see Figure 3. The difference between Procedure Steps and Process Steps is that Procedure Steps wait for the call to complete before the application can proceed while Process Steps do not.

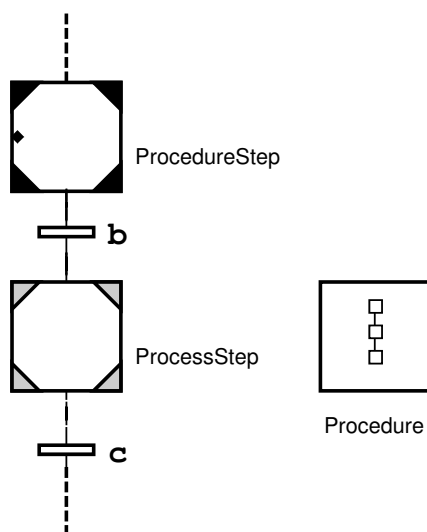


Figure 3: A Procedure can be called from Procedure Steps and Process Steps. Each Procedure Step and Process Step specify which Procedure to call when activated.

### 3.1 Execution Model

Grafchart has a well defined execution model which ensures sufficiently deterministic execution behavior. A transition is *enabled* when all immediately preceding steps are active. An *enabled* transition is *fireable* if

its condition is true. *Firing* a transition involves deactivating the immediately preceding steps and activating the immediately succeeding steps.

Grafchart applications are, like SFC, executed periodically, one scan cycle at a time. The execution model of a scan cycle is described by the following sequence:

1. Read inputs.
2. Mark fireable transitions.
3. Remove mark for conflicting transitions of lower priority.
4. Fire marked transitions.
5. Update step properties  $t$  and  $s$ .
6. Execute P actions.
7. Mark variables subject to N actions.
8. Update marked variables.
9. Sleep until the start of the next scan cycle.

The execution model has the property that an activated step always remains active for at least one scan cycle. Note that the execution model does not give a completely deterministic execution. For example the firing order of transitions affects which step's S and X actions are executed first. Another example is which step's P actions are executed first. The application is not allowed to depend on the execution order in these cases.

## 3.2 JGrafchart

JGrafchart is a free Java based integrated development environment for the Grafchart programming language which can be downloaded from <http://www.control.lth.se/Research/tools/grafchart.html>. It is a research tool used in for example the EU/GROWTH project CHEM for control in process industry [5], the EU FP7 project ROSETTA for robotic assembly [6], and a master's thesis for modeling of avionics systems [7].

### 3.2.1 Inputs and Outputs

JGrafchart can be connected to external environments through a multitude of customizable input/output (I/O) integration capabilities and can thus be used to control external real and simulated processes. This is used in education, for example in laboratory exercises on sequential and batch control, and to control real industrial processes.

One I/O possibility in JGrafchart is the *CustomIO*, that is, the I/O elements Digital In, Digital Out, Analog In, and Analog Out as well as inverted variants

for the Digital In/Out. At the beginning of each scan cycle each In I/O is read from the external environment. An Out I/O is written to the external environment whenever assigned. How the I/O interact with the external environment depends on the chosen I/O implementation. A custom I/O implementation is created by implementing a set of Java interfaces. With a custom implementation it is possible to communicate with practically any external environment. However, it is limited to Boolean and Real values.

There is also generic support for communicating with Devices Profile for Web Services (DPWS) devices using the DPWS4J toolkit [8]. Devices and their supported operations are automatically discovered and can be called directly.

Another I/O possibility is the *SocketIO* elements. JGrafchart then connects to a TCP server and communicates Boolean, Real, Integer, and String values over a socket with the message protocol: `<identifier> '|> <value> '\n'`. The TCP server is responsible for the interaction with the external environment. *SocketIO* is often powerful enough to allow creation of external adapters to other communication protocols. It has for example been used to integrate JGrafchart with a multitude of tools and protocols, among others prototypes for Simulink [7], DPWS [9], LabComm, and OPC UA support.

It would be useful to also support code generation to be able to export JGrafchart applications as FMUs. Code generation has previously been added to JGrafchart [10, 11] but, due to the current JGrafchart code base design, the results have been limited and fragile.

Currently JGrafchart only supports interpreted execution. To execute an application it must first be compiled. The compiler checks if the application is valid and prepares it for execution by attaching additional data. Applications are then executed directly in an interpreted manner using the same Java instances as the editor. JGrafchart is currently being split into three standalone parts, namely editor, compiler, and executor. This makes it possible to add robust code generation capabilities.

## 4 JGrafchart with FMI Support

### 4.1 Motivation

In one of our laboratory exercises, JGrafchart is used to control both a simulated and a real batch tank. The simulated process is implemented as a simpli-

fied model in Java. It is also possible to implement simulations of simple processes directly in JGrafchart [12]. However, there is much potential for improvement in terms of effort for specifying the simulated model, quality of the models, support for inspecting simulation results, and time required to simulate, especially for more complicated physical systems. Extending JGrafchart with support for *FMI for Co-Simulation* gives more and better opportunities to connect JGrafchart to other tools.

There is also a need to efficiently develop and test JGrafchart control applications before using them to control the real system. This may save a lot of time as many industrial systems have slow dynamics and running a simple test on the real system could take days. With a good model of the system the development time could be considerably reduced, and the quality of the control application will be higher as there is less resistance in the development process. Industrial systems are often dangerous and running a proper simulation first could be essential for safety reasons. For the batch tank in our laboratory exercise, the simulated process is 10 times as fast as the real process. Special code is required to add support for this which both makes it fragile and susceptible to errors as it is possible to run the control application in the wrong mode, for example in simulation mode against the real system. With a simulation environment that does not run according to wall clock time, it can be run faster and without the special code.

It is also important to verify that the system behaves properly when controlled by a JGrafchart application. JGrafchart executes periodically and only sees the sampled behavior. When controlling a continuous system, the behavior between the sampling points may also be of interest. Also, currently JGrafchart applications are always executed according to wall clock time. With a large or complex simulated system the JGrafchart application might execute faster than the rest of the system can be simulated.

Support for state machines were introduced in Modelica 3.3 [13] providing a proper way to implement hierarchical state machines directly in Modelica. On one hand, JGrafchart does not provide the mutual hierarchical structuring property with data flow that Modelica state machines do [13], but on the other hand it supports powerful high level language features such as object orientation, hierarchical structuring, code reuse, and exception handling. Additionally it is based on an industrial automation language.

## 4.2 Integration

JGrafchart supports the data types Real, Integer, Boolean, and String which correspond to the FMI data types `fmiReal`, `fmiBoolean`, `fmiInteger`, and `fmiString`. Both variables, lists (arrays), and I/O in JGrafchart use only these data types. The state of a JGrafchart application is described by the variable, list, and I/O values as well as which steps are active, how long they have been active, and currently active procedure calls. The number of simultaneous procedure calls and the list sizes are not limited and there are no semantics to limit this. However, there are no list I/O and procedures are not allowed to contain I/O so this is not an issue for *FMI for Co-Simulation*.

As the I/O in the JGrafchart application are the means of connecting it to external components, these would ideally be the FMU inputs and outputs. The mapping for *CustomIO* and *SocketIO* is straightforward. DPWS on the other hand is based on method calls instead of data and need some configuration to be able to expose the methods as data instead. Thus it is best to exclude DPWS, at least during prototyping.

JGrafchart applications are executed periodically, one scan cycle at a time, as described in Section 3.1. The execution can be modeled as discrete events at the beginning of each scan cycle. During the rest of the scan cycle nothing happens. Ideally, there would be *communication points* just before and just after the beginning of each scan cycle. With a sufficiently small *communication step* size this should work fine for all JGrafchart applications, regardless of scan cycle time. This could be requested by setting the `stepSize` attribute of the `DefaultExperiment` element in the FMU XML.

JGrafchart applications are currently always executed according to wall clock time and it is not possible to get and set the execution state as there has been no need for this before. However, it should be possible to extend JGrafchart with the possibility to get and set the current execution state to support redoing *communication steps*.

## 4.3 Architecture

This section discusses various ways to connect a JGrafchart application to an FMI *master*.

### 4.3.1 Hardware-in-the-loop

The simplest way is to consider the JGrafchart application as a hardware-in-the-loop, see Figure 4. Then the

JGrafchart application executes as usual, with the FMI *master* getting and setting its I/O. As discussed before, the co-simulation must then be able to keep up with and synchronize with JGrafchart's wall clock time execution. The main advantage with this approach is that no modifications to JGrafchart are necessary, it would be sufficient to create an FMU compatible *CustomIO* or TCP server for *SocketIO*. This is a suitable approach for FMU integration prototyping but it does not improve matters for systems with slow dynamics.

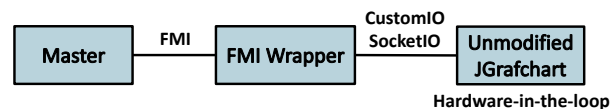


Figure 4: Overview of connecting JGrafchart as a hardware-in-the-loop.

### 4.3.2 Generic FMI Wrapper

Another approach is to implement a generic FMI wrapper for JGrafchart and extend JGrafchart with support for external clocks, see Figure 5. It is a small effort to add this feature. The same FMI wrapper would be possible to use with all JGrafchart application but the wrapper would expose different inputs and outputs to the FMI *master* depending on the JGrafchart application. This approach only requires slightly more effort than the hardware-in-the-loop approach and gives more benefits as the co-simulation no longer executes according to wall clock time. For this approach it is suitable to also add support for playback and to be able to inspect individual scan cycles of the JGrafchart application during the co-simulation. To add these features should only be a moderate effort, it could be as simple as trace printouts or as advanced as interactive scan cycle stepping. Compared to the hardware-in-the-loop case, the main drawback is that modifications to JGrafchart are required. However, these additions are great additions in general and are not solely useful for *FMI for Co-Simulation*. For example they open up possibilities for integration with other tools and improves JGrafchart's debugging capabilities.

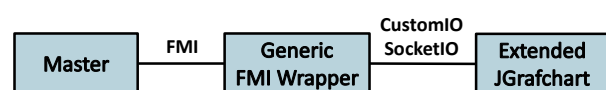


Figure 5: Overview of connecting JGrafchart with a generic FMI wrapper.

### 4.3.3 Standalone FMU

The last approach discussed in this paper is to generate a standalone FMU for a JGrafchart application, see Figure 6. The FMU is then self-contained and does not rely on JGrafchart running in parallel. This is a clean and portable approach but requires the most effort and might make it harder to inspect the co-simulation results. Until the JGrafchart compiler is standalone, the implementation would also be fragile.

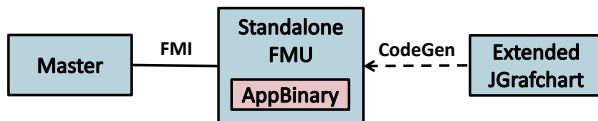


Figure 6: Overview of using code generation to create a standalone JGrafchart FMU.

A hybrid approach is to use a generic FMI wrapper with both JGrafchart and the JGrafchart application embedded as additional FMU *resources*, see Figure 7. Then the FMU is standalone but no code generation is required. The main drawback with this approach is that the FMUs would be roughly 20 MB larger.

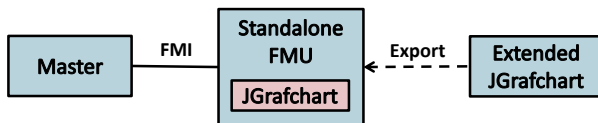


Figure 7: Overview of exporting a standalone FMU with integrated JGrafchart and JGrafchart application.

## 4.4 Implementation

The FMI API is defined for C and FMUs are distributed with C source code and/or binary executables for supported platforms. JGrafchart is written in Java and is platform independent. However, the FMU itself can be implemented in any language which is able to interact with C code, that is, practically any language. There are language specific wrappers for FMI, for example PyFMI for Python [14] and JFMI for Java [15] which uses JNA [16] to interface with native code. Which language is chosen is less important and up to the one who implements the FMU. However, the *CustomIO* implementation must be written in Java.

## 5 Future Work

Extending JGrafchart to support *FMI for Co-Simulation* is only conceptual so far. It looks promis-

ing and there are several alternative ways that it could be implemented.

The next step is to implement a prototype to verify that it works in practice. A suitable first attempt would be a hardware-in-the-loop approach using an unmodified version of JGrafchart and utilizing its *CustomIO* and/or *SocketIO* capabilities. However, this does not improve matters for slow systems. A desirable future solution would either be the Generic FMI Wrapper or the Standalone FMU.

## Acknowledgements

The authors are members of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University.

## References

- [1] IEC. *IEC 61131-3: Programmable controllers – Part 3: Programming Languages*. Tech. rep. International Electrotechnical Commission, 1993.
- [2] Alfred Theorin. *Adapting Grafchart for Industrial Automation*. Licentiate Thesis ISRN LUTFD2/TFRT--3260--SE. Department of Automatic Control, Lund University, Sweden, 2013-05.
- [3] FMI Development Group. *Functional Mock-up Interface for Model Exchange and Co-Simulation – 2.0 Release Candidate 1*. Tech. rep. Modelica Association, 2013-10.
- [4] Modelica Association. *Modelica*. URL: <https://www.modelica.org/> (visited on 2013-12-08).
- [5] Karl-Erik Årzén, Rasmus Olsson, and Johan Åkesson. “Grafchart for Procedural Operator Support Tasks”. In: *Proceedings of the 15th IFAC World Congress, Barcelona, Spain*. 2002-07.
- [6] Andreas Stolt. *Robotic Assembly and Contact Force Control*. Licentiate Thesis ISRN LUTFD2/TFRT--3256--SE. Department of Automatic Control, Lund University, Sweden, 2012-12.
- [7] Anna Benktson and Sofia Dahlberg. *Modeling of Avionics Systems using JGrafchart and True-Time*. Master’s Thesis ISRN LUTFD2/TFRT--5907--SE. Department of Automatic Control, Lund University, Sweden, 2012.

- 
- [8] SOA4D Forge. *DPWS4J Core*. URL: <https://forge.soa4d.org/projects/dpws4j/> (visited on 2013-12-07).
- [9] Alfred Theorin, Lisa Ollinger, and Charlotta Johnsson. “Service-oriented Process Control with Grafchart and the Devices Profile for Web Services”. In: *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics*. Ed. by Theodor Borangiu, Andre Thomas, and Damien Trentesaux. Vol. 472. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2013-01, pp. 213–228. ISBN: 978-3-642-35851-7. DOI: [10.1007/978-3-642-35852-4\\_14](https://doi.org/10.1007/978-3-642-35852-4_14). URL: [http://dx.doi.org/10.1007/978-3-642-35852-4\\_14](http://dx.doi.org/10.1007/978-3-642-35852-4_14).
- [10] Isolde Dressler. *Code Generation from JGrafchart to Modelica*. Master’s Thesis ISRN LUTFD2/TFRT--5726--SE. Department of Automatic Control, Lund University, Sweden, 2004-03.
- [11] Ana Llorente. *Code Generation from JGrafchart to ATMEL AVR*. Master’s Thesis ISRN LUTFD2/TFRT--5749--SE. Department of Automatic Control, Lund University, Sweden, 2005-01.
- [12] Alfred Theorin and Charlotta Johnsson. “An Interactive PID Learning Module for Educational Purposes”. In: Submitted to *The 19th World Congress of the International Federation of Automatic Control (IFAC)*. 2014-08.
- [13] Hilding Elmqvist et al. “State Machines in Modelica”. In: *Proceedings of 9th International Modelica Conference, Munich, Germany, September*. 2012, pp. 3–5.
- [14] JModelica.org. *PyFMI*. URL: <http://www.jmodelica.org/page/4924> (visited on 2013-12-08).
- [15] The Regents of the University of California. *JFMI - A Java Wrapper for the Functional Mock-up Interface*. URL: <http://ptolemy.eecs.berkeley.edu/java/jfmi/index.htm> (visited on 2013-12-08).
- [16] Todd Fast, Timothy Wall, Liang Chen. *Java Native Access (JNA)*. URL: <https://github.com/twall/jna> (visited on 2013-12-08).





# Development of Custom Workflows for Simulation and Analysis of Functional Mock-up Units

Sureshkumar Chandrasekar<sup>1</sup>

<sup>1</sup>Modelon Inc.  
Hartford, CT  
United States

[chandrasekar.sureshkumar@modelon.com](mailto:chandrasekar.sureshkumar@modelon.com)

Jesse Gohl<sup>2</sup>

<sup>2</sup>Modelon Inc.  
Ann Arbor, MI  
United States

[jesse.gohl@modelon.com](mailto:jesse.gohl@modelon.com)

## Abstract

Development of customized workflows and interfaces to deploy Modelica Functional Mockup Units (FMUs) with the various FMU tools has been gaining traction in the industry – both with tool vendors as well as end users.

The FMI Add-in for Excel (FMIE) is a commercial product from Modelon AB that enables the deployment of FMUs in Microsoft Excel. FMIE enables the user to programmatically control the add-in through Visual Basic code in Excel. This allows the implementation of custom workflows and interfaces for the user to interact and automate the tasks involved in loading, simulation and analysis of FMUs.

In this paper we present a workflow in FMIE for an automobile thermal management model FMU. The workflow utilizes Visual Basic scripts for automation and user-forms for user interaction.

*Keywords: FMU; Automated workflow; FMI Add-In for Excel; Visualization; Design of Experiments, Batch Simulations, Monte-Carlo Analysis*

## 1 Introduction

The Functional Mock-up Interface (FMI) defines an interface to be implemented by an executable called the Functional Mock-up Unit (FMU) to describe, evaluate and simulate models of dynamic systems defined by differential, algebraic and discrete equations in different simulation environments [1][2]. The FMI standard has greatly facilitated the transfer, exchange of dynamic models between component suppliers and Original Equipment Manufactureres (OEMs) and enabled export and import of models between multiple simulation platforms that have FMI export/import functionality.

The FMI standard as a result, is gaining traction in the industry as the technology of choice in the ex-

change of dynamic models between suppliers and OEMs. As a result, there have been a large number of FMU deployment tools of which the FMI Add-In for Excel (FMIE) [3] developed by Modelon AB is a popular choice among customers.

The FMIE integrates FMI-based parallel simulations in Excel. The add-in offers the following features:

- Simulation of compiled dynamic models, FMUs generated by any FMI-compliant tool such as Dymola, OPTIMICA Studio or Simulation X.
- The add-in takes advantage of powerful features in Excel to set up and perform batch simulations for parameter sweeps and simulations driven by data series.
- Perform dynamic simulations or solve initialization problems, in parallel.
- The FMI add-in supports FMU for Model Exchange 1.0 and FMI for Co-Simulation 1.0.

Due to its application in multiple engineering domains and across several organizations, FMIE provides a general interactive environment through the Microsoft Ribbon interface. From version 1.2 onwards, FMIE enables the user to programmatically control the add-in through Visual Basic code in Excel. This has opened up interesting possibilities to create custom workflows to automate several tasks in the loading, simulation and analysis of FMUs in Excel with FMIE.

The workflow includes both scripting automation and a customized graphical user interface (GUI). The automation includes filtering of variables to set-up experiments, loading values for these filtered variables for simulation and scripts for plotting the output response to these experiments. Additionally, macros to do the same for Design of Experiments (DoE) and Monte-Carlo analysis where many hundreds of simulation cases are to be generated in a single experi-

ment possibly with statistical distributions for multiple parameters/variables. The workflow presented in this paper allows all of these steps to be performed in just a few mouse clicks interactively and improves efficiency and ease of use.

Further, for visualization of results, a customized GUI is presented as a Visual Basic for Applications user form within Excel. It presents a streamlined interface for plotting experiment trajectories to the end user. This allows the user to quickly find available trajectories and generate default charts for visualizing and comparing the results.

## 2 The FMIE Interface

The FMIE provides a general graphical interface through the Microsoft Ribbon interface and appears as one of the tabs in the Ribbon interface of Excel and labeled as 'FMI' as shown in Figure 1. The FMI tab contains four different groups of buttons.

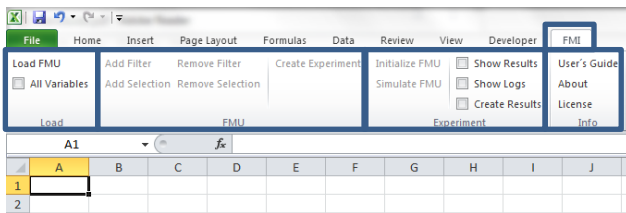


Figure 1: The FMIE Native Interface

The FMI tab in the ribbon is divided into four groups: **Load**, **FMU**, **Experiment** and **Info** as highlighted in Figure 1 and respectively have buttons to load, create and set-up, simulate and analyze results of FMU's. In addition to these buttons, the functionalities of these buttons are mirrored with API function calls that are listed in Table 1. These functions are written in Visual Basic and the code for these is accessed from opening Visual Basic from the Developers tab.

FMIGetVersion	FMIUpdateSheet
FMIlicenseCheckedOut	FMIGetCreateResults
FMIsetCreateResults	FMIGetShowResults
FMIsetShowResults	FMIGetShowLogs
FMIetShowLogs	FMIcreateExperiment
FMIloadFMU	FMIsetFilterButton
FMIsetSelectionButton	FMIsimulateButton
FMIinitializeButton	FMIsimulate0
FMIinitialize0	FMIsimulate1
FMIinitialize1	

Table 1: List of API functions in FMIE.

### 2.1 Structure of loaded FMU and experiment sheets

The FMIE follows a pre-defined structure to how the information about input, output variables and parameters from the FMU are displayed in Excel. The loaded FMU displays meta-data containing general information of the FMU model e.g. model name, FMU kind, number of state variables etc. The list of input, output variables and parameters are displayed under the Variables section in the same worksheet. The variables are identified by the name, variability, start value, unit, description etc. and there is a column to indicate if the variable in the corresponding row is to be included in an experiment. This Boolean is set to "TRUE" if the variable is to be selected for analysis in an experiment. Each of these columns has a search filter associated with which the variable list can be parsed to select the appropriate variable set for analysis. This process gets tedious and inefficient when the number of variables in an FMU or the number of variables to be selected in an analysis is huge. As a result, any workflow to be developed should provide an easy and efficient way to filter out these variables for analysis.

Once the analysis variables are filtered and an experiment sheet created, the input variables and parameters are separated automatically by FMIE from the output variables in the experiment sheet. The nominal values of the inputs and parameters are also displayed and 3 default simulation cases already set-up. The user is to then create as many simulation cases as needed within the experiment by creating new columns adjacent to the 3 already created. The next step before simulating the FMU is to load the parameter, input values for the experiment. This is also to be done manually in the native interface and can be automated to improve efficiency. The next section describes a template for entering the variable list and parameter values for experiments that would enable the workflow created to automatically load these into the experiment sheet.

## 3 Workflow

A typical workflow involved in the set-up, simulation and analysis of FMUs is illustrated in Figure 2. At a higher level, the workflow follows a sequential procedure to load the FMU, filtering the relevant input, parameter and output variables for the study, create experiments and loading the input data for simulation, plotting and analysis of results.

To standardize the interface for user input and to facilitate data exchange between model developers and end-users, we present a simple front-end template for user input of information or data pertaining to the analysis to be performed. We present two templates – one for batch simulations involving parameter sweeps and the other for DoE and Monte Carlo studies. The proposed workflow is demonstrated in this paper with the example of a Co-Simulation FMU of a automotive thermal management system proposed in [4].

### 3.1 Batch Simulations

Batch simulations include parameter sweeps and experiments where one or more variables are changed individually or simultaneously and studying the effects of these on certain output variables. As a result, the batch simulation sheet needs to include sections for including the names of the input, parameter and output variables along with a description of each variable. Columns for entering user data are then classified according to the names of the experiments as shown in Figure 3.

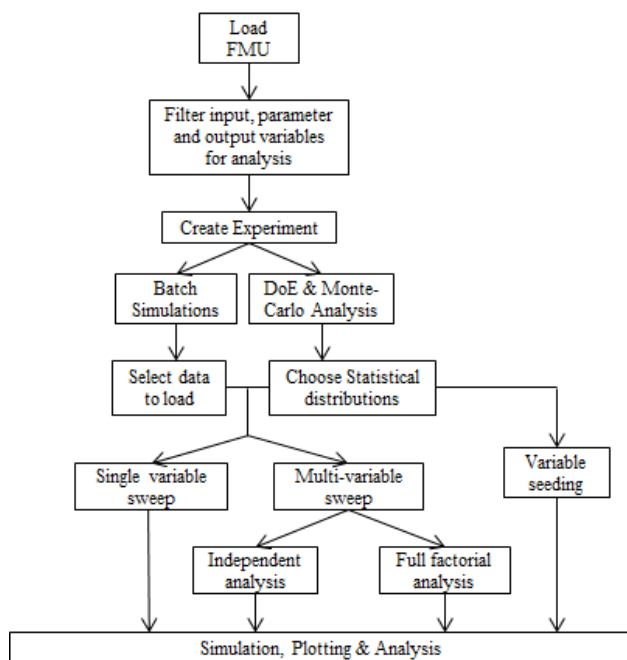


Figure 2: A typical workflow in the set-up, simulation and analysis of an FMU.

### 3.2 Design of Experiments and Monte-Carlo Analysis

Similar to the batch simulation template, a template for Monte-Carlo analysis is also set-up as shown in Figure 4. This template includes fields to enter the type of statistical distribution and arguments for each statistical distribution. Figure 4 shows an example

where the efficiency multiplier of the heat exchanger stack is varied according to a Gaussian distribution with mean 1 and standard deviation 0.1. The current implementation of the workflow supports Gaussian, uniform and triangular distributions and can be easily extended to other Excel-supported statistical distribution functions.

Both templates provide a standardized structure for user-input of data for the workflow and enable convenient exchange of information between model developers and end users of the FMU.

Additional fields like the base class have been included in the template for model developers to provide a short description or a snapshot of the base class in which the variable is used, so as to give the end-users some insight into the variable.

### 3.3 Load FMU and Filter Variables

With the batch simulation or DoE template filled up with a set of variables and their corresponding input data like that shown in Figure 3-4, the next step in the workflow is to filter these variables and create an experiment sheet. A sample (partial) Visual Basic code shown in Table 2 is an example that loads an fmU and prompts the user to choose variables for filtering and creates an experiment with 7 cases with these variables.

```

Public Sub FMIE_Filter_vars()
Dim fmuFile As String
On Error GoTo ErrFileNotFound
' Path to FMU
fmuFile = Application.ActiveWorkbook.Path &
"\DriveCycleVTM.fmu"
' Load FMU into Excel
FMILoadFMU fmuFile, True, vbNullString
' Select & Filter variables to create experiment
Dim data As Range
Set data = Application.InputBox(Prompt:=
"Please select range", Title:="Range Select",
Type:=8)
<<Add VB code to set the Include attribute of
selected data to True>>
' Create Experiment sheet with 7 simulation cases
FMICreateExperiment ActiveSheet,vbNullString ,7
Exit Sub
ErrFileNotFound:
MsgBox "Error: FMU not found!"
  
```

Table 2: Sample VB code for loading FMU and filtering variables

The above code loads the fmU named 'DriveCycleVTM.fmu', and then prompts the user to select the names of the variables from the batch simulation or the DoE template sheet to be added to an experiment.

Enter Data in this sheet to perform Batch Simulations on FMU's				Modelon															
<b>Instructions:</b> 1. Load the FMU 2. Enter data for batch simulations for the parameters/variables of interest into the respective rows. <i>Max Limit: 16372</i> entries per variable. 3. Ensure the naming convention is consistent between this & the FMU sheets. 4. Set up the Experiment sheet by filtering the variables/parameters of interest. 5. Call the LoadData macro from the <i>experiment sheet</i> and use the user form to choose the variables and data entered in this sheet. For a full factorial sweep of all variables, check the permute option. 6. The data are now loaded in the experiment sheet and the experiments are ready to be simulated using FMIE.																			
Inputs	Base Class	Description	Variable/Parameter	Default	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	Sim. 7	Default	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	
		Inclination in the x direction	driverVehicle.ground_k_x	0									0						
		Final gear ratio rear	driverVehicle.vehicle.driveline.rear_ratio	4									4						
		CG height above line connecting wheel centers	driverVehicle.vehicle.h_cg	0.3									0.3						
		Vehicle body mass	driverVehicle.vehicle.body_mass	1600									1600						
		Undeformed radius - front wheel	driverVehicle.vehicle.R0_front	0.3									0.3						
		Undeformed radius - rear wheel	driverVehicle.vehicle.R0_rear	0.3									0.3						
		Inertia about the spin axis - front wheel	driverVehicle.vehicle.wheel_L_xi_front	1									1						
		Inertia about the spin axis - rear wheel	driverVehicle.vehicle.wheel_L_xi_rear	1									1						
		Aerodynamic drag coefficient of car body	driverVehicle.vehicle.c_w_front	0.38									0.38						
		Frontal area of car body	driverVehicle.vehicle.a_front	2.7									2.7						
		scaling factor for the drivecycle velocity	driverVehicle.driveCycleNameIndex	2	23	11	9	16	19	18	15		3						
		Ambient temperature (inlet to the stack)	driverVehicle.driveCycleScale	1									1	10	15	20	25	30	35
	low fan speed temperature threshold	Tamb	311									311							
	medium fan speed temperature threshold	simpleInletConditions.airFlowGain	1									1							
	high fan speed temperature threshold	fanController.loSpeedThresh	360									360							
	max fan speed temperature threshold	fanController.medSpeedThresh	380									380							
	close grill when speed exceeds this threshold	fanController.hiSpeedThresh	400									400							
	open grill when speed falls below this threshold	fanController.maxSpeedThresh	430									430							
	grill effect air flow tuning gain	grillController.closeThreshold	18									18							
	drive cycle	grillController.openThreshold	15									15							
Outputs	Base Class	Description	Variable/Parameter	Experimental Results (if any) for the cases above															
		Value of Real output	signalBus.VDL_stack_Tair_in_1																
		Value of Real output	signalBus.VDL_stack_Tair_in_2																
		Value of Real output	signalBus.VDL_stack_Tair_in_3																
		Value of Real output	signalBus.VDL_stack_Tair_in_4																
		Value of Real output	signalBus.VDL_stack_Offlow_1																
		Value of Real output	signalBus.VDL_stack_Offlow_2																
		Value of Real output	signalBus.VDL_stack_Offlow_3																
		Value of Real output	signalBus.VDL_stack_Offlow_4																
		Absolute velocity vector of front wheel	signalBus.VDL_veh_vel_x																
		Connector of Real output signal	simpleFan.fanSpeed																
			signalBus.VDL_eng_cpd																
			signalBus.VDL_eng_fra																
		signalBus.VDL_eng_coolTemp																	

Figure 3: Template for user-input for batch simulations in FMIE.

Enter Data in this sheet to perform Monte-Carlo Simulations on FMU's				Modelon											
<b>Instructions:</b> 1. Load the FMU 2. Enter data for monte-carlo simulations for the parameters/variables of interest into the respective rows. <i>Max Limit: 16370</i> entries per variable. 3. Ensure the naming convention is consistent between this & the FMU sheets. 4. Set up the Experiment sheet by filtering the variables/parameters of interest. 5. Call the MonteCarloData macro from the <i>experiment sheet</i> and use the user form to choose the number of points, the statistical parameters for the uncertain variables and data for other constant variables. Only Normal/Uniform distributions are supported for now.															
Inputs	Base Class	Description	Variable/Parameter	Distribution	Arg 1	Arg 2	Arg 3	Arg 4	Comments	Note					
		Efficiency multiplier	stack.heat_exchanger_1_eff_multiplier	Gaussian	1	0.1			Arg 1 is mean and Arg 2 is standard deviation	Uniform (or flat), Gaussian (or normal) and Triangle distributions are supported. For uniform, arg1 and arg2 are used as the min & max values. For gaussian, arg1 is mean and arg2 is std-dev. For triangular, arg1, arg2, arg3 are min, max and peak values					
		Efficiency multiplier	stack.heat_exchanger_2_eff_multiplier	Gaussian	1	0.1									
		Efficiency multiplier	stack.heat_exchanger_3_eff_multiplier	Gaussian	1	0.1									
		Efficiency multiplier	stack.heat_exchanger_4_eff_multiplier	Gaussian	1	0.1									

Figure 4: Template for user-input for DoE and Monte Carlo analysis in FMIE.

The number of simulation cases to be set-up in the experiment sheet is also stated explicitly as shown in Table 2.

### 3.4 Loading Data into an Experiment

As noted in 3.1 above, the batch simulation template sheet includes fields to include the data for inputs and parameters to be set in experiments. Figure 3 includes data for two experiments labeled ‘Exp DC Batch’ and ‘Exp Const Speed’ where the model is simulated with different drive cycles in the former and a sweep over different speeds with a constant speed drive cycle in the latter.

To allow a more general way for users to load input data into the experiment sheets involving multiple parameter sweeps in the same experiment either as a full factorial or independent study, the user form shown in Figure 5 is used. A similar form for data input for Monte-Carlo Analysis is shown in Figure 6.

These user forms contain private sub-routines for processing the selected input and based on the options chosen, load the input data into the experiment sheet.

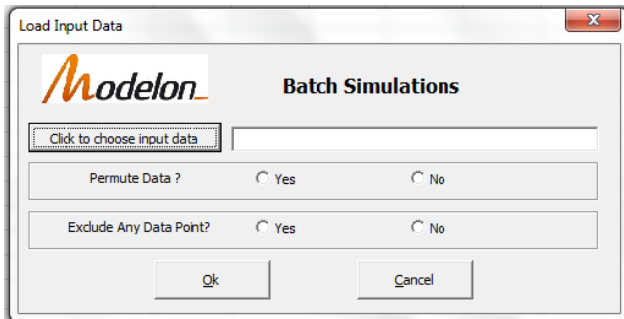


Figure 5: User form for data input for batch simulations

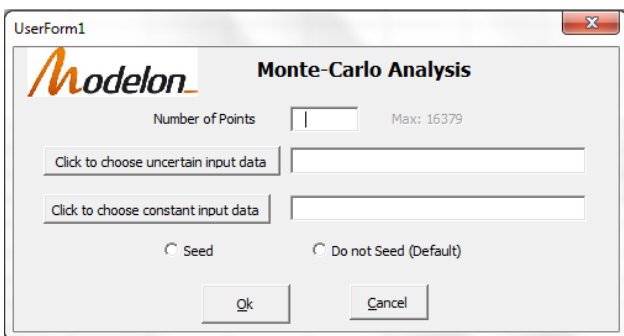


Figure 6: User form for data input for Monte Carlo analysis.

The user form for batch simulation for example lets the user choose a range of values for each variable and gives the option to run a full factorial sweep of

all values for all the parameters or input variables and also the option to choose a particular set of data points for exclusion from simulations. The user form for Monte-Carlo analysis has fields to enter the number of simulations cases to be performed in the study and options to choose the uncertain parameters and their corresponding attributes (statistical distributions and the corresponding arguments) and also the option to set input data for variables that remain constant throughout the simulation study. The option to seed gives the user the ability to repeat the simulation by generating the same set of randomly generated points as from a prior experiment.

The userforms described in this section along with the private subroutines that process the user input and load the data into the FMIE experiment sheet can be combined with the VB code shown in Table 2 to seamlessly integrate the entire FMU and experiment set-up procedure into a single macro for execution. And when combined with the analysis tools for plotting that are described in the next section, provide the user with an efficient and easy-to-use workflow in working with FMUs.

### 3.5 Plotting simulation results

When an end user chooses the option ‘‘Create Results’’ from an experiment sheet, FMIE will create a new ‘‘Result’’ worksheet. In general this will be the same name as the experiment sheet with the prefix ‘‘Res’’. The structure of these result worksheets are column oriented data of the selected output trajectories sequentially ordered by experiment case. From there the user is free to generate charts of this data as they would with other Excel data. To further increase usability for end users, custom GUIs can also be created to manage and generate these charts. An example of this is shown in Figure 7.

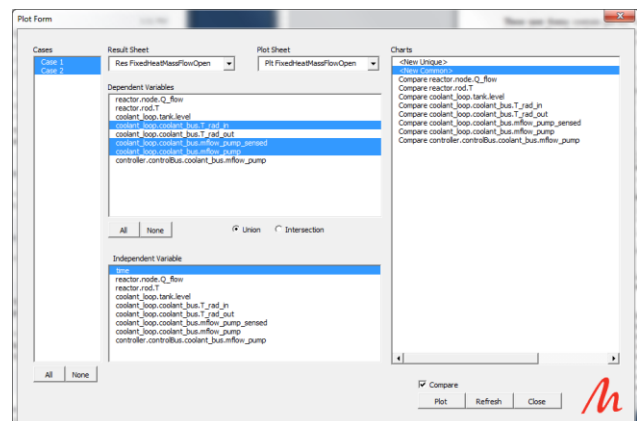


Figure 7: Illustration of the plotting GUI

The GUI maintains multiple lists relevant to plotable data from experiments. The first is a list of re-

sult sheets within the workbook. This allows the user to select which set of results they wish to use. When a particular result sheet is selected, the “Cases” list is updated with all the cases within the result sheet.

The user is then free to choose any combination of cases they wish to work with. As the case list selection changes the lists of dependent and independent variables also changes to match the corresponding variables for the selected cases. The user also has the option of choosing either the union or intersection of the set of variables for each selected case.

The user then chooses any combination of dependent variables and a single independent variable. By default time is selected. Next the user can choose an existing “Plot Sheet” or define a new sheet directly within the drop down list. The Chart list is updated to reflect existing charts on the sheet. This allows the user to add the selected trajectories to existing charts or to create new charts with the selected signals.

When new charts are chosen they are automatically added to the sheet without needing any user intervention on placement or style. The user is however free to manually modify the chart properties after it is created.

## 4 Conclusions

In this paper, an approach for implementing a workflow to deploy FMUs with the FMI Add-In for Excel was presented. A typical workflow in the set-up and analysis of FMUs was presented along with a general template in Excel of a front-end with which the end-user interacts. The macros created with the user forms for loading data and the FMIE API functions can be used as the back-end which enable the end-user to quickly and efficiently simulate and analyze FMU models with relative ease. They can also be used as the front-end interface for generating charts of simulation data for quick and simple visualization of results.

## References

- [1] Modelisar ITEA2, “Functional Mock-up Interface for Model Exchange”, Accessed from [www.fmi-standard.org](http://www.fmi-standard.org) in Nov 2013.
- [2] Modelisar ITEA2, “Functional Mock-up Interface for Co-Simulation”, Accessed from [www.fmi-standard.org](http://www.fmi-standard.org) in Nov 2013.

- [3] Modelon AB, “FMI Add-In for Excel”, Accessed from <http://www.modelon.com/products/fmi-add-in-for-excel/> in Nov 2013.
- [4] Batteh J., Gohl, J., S Chandrasekar, “Integrated Vehicle Thermal Management in Modelica: Overview and Applications” *Proceedings of the 10<sup>th</sup> International Modelica Conference*, Lund, 2014.

# A Medium Model for the Refrigerant Propane for Fast and Accurate Dynamic Simulations

Roozbeh Sangi

Pooyan Jahangiri

Freerk Klasing

Rita Streblov

Dirk Müller

Institute for Energy Efficient Buildings and Indoor Climate,  
E.ON Energy Research Center, RWTH Aachen University,  
Mathieustr.10, 52074 Aachen, Germany  
rsangi@eonerc.rwth-aachen.de

## Abstract

Investigating the use of different fluids and their advantages in new energy systems has increased the need for faster and more robust simulation models. The need to explore the potential of new fluids in different systems requires dynamic simulations for longer periods of time. In this paper, developing and improving a medium model for propane is discussed. Besides being fast and accurate, the propane model should also be stable in different dynamic simulation scenarios.

First, existing libraries are tested and in some cases modified to increase the stability. Since the simulation speeds were not in an acceptable range in the existing models, a new propane model based in the refrigerant models in HelmholtzMedia Library is introduced. The new model is then tested as a refrigerant in a direct exchange heat pump system. A comparison between an existing propane model and the new model shows that much faster simulations, up to 35 times, are possible with the new propane model.

*Keywords: Media Library; Propane; Refrigerant; Helmholtz Media*

## 1 Introduction

To address the main challenges arising from an ever increasing energy demand worldwide and its associated environmental impacts, it is not only essential to optimize the existing energy landscape but is also necessary to develop new approaches. One of these approaches is to utilize low-exergy heating and cooling systems.

A well-known method is by the use of heat pumps to extract energy stored in the ground. A conventional combination of a heat pump and a heat-

ing/cooling unit consists of three different hydraulic cycles, a primary cycle (energy source), a secondary cycle (energy sink) and a refrigerant cycle in between. For dynamic analysis of a complete heating or cooling system using a heat pump, the refrigerant cycle is usually considered as a black box model. This means that the thermal behavior of the refrigerant under different conditions is considered to be known without knowing the actual state of the refrigerant. Although this will reduce the accuracy of such a simulation, in larger scales, its great impact in the simulation speed is of higher importance.

To improve the affordability of heat pumps and reduce thermal losses in the heat exchangers between each cycle, the concept of direct exchange heat pumps is considered. This means that instead of having a heat exchanger between the energy source (ground) and the refrigerant cycle, the refrigerant flows directly inside the ground source. In simulations of such systems, a black-box model cannot be used and the whole behavior of the refrigerant should be simulated without compromising the simulation speed.

There are different methods of simulating different refrigerants. C. Heinrich et al. [1] have developed models for household refrigerant appliances using R600a, T. Pfafferott et al. [2, 3] have developed refrigerant models with CO<sub>2</sub> and R134a and I. Bell et al. [4] have created an open-source fluid library available for many different platforms using the Helmholtz equation of state. These methods use either internal or external functions. Although these methods can be expanded to different refrigerants, a separate propane model is not developed and the accuracy of the models was the main subject and not the simulation speed. Since simulation speed is of great concern, external media functions are not considered in this study. In the next chapter an available method of describing the behavior of refrigerants

internally is discussed. For further simplifications, only one refrigerant is considered.

Because of its low environmental impact, propane is chosen as the refrigerant and the modeling of this refrigerant is described in this paper.

## 2 Available Refrigerant Model

The design and simulation of heat pumps or power cycles require an accurate representation of the working fluid. In general a fluid is described by the equation of state and additional transport properties. The equation of state defines all thermodynamic properties in terms of two independent thermodynamic state variables. Usually these two variables are pressure ( $p$ ) and enthalpy ( $h$ ), temperature ( $T$ ) and density ( $\rho$ ) or, in case of one phase only, pressure and temperature. Today fundamental equations of state (EoS) in terms of Helmholtz energy are the most accurate method available for this purpose [5].

### 2.1 Helmholtz Energy Equation of State

As described in detail by [6], the Helmholtz EoS uses temperature and specific volume or density of the fluid as the independent variables. Other variables such as specific enthalpy, specific entropy as well as partial derivatives of several thermodynamic state variables can be calculated directly from these independent variables. Additional ancillary equations express the vapor pressure and the saturated densities in terms of the temperature and thereby define the boundary between single-phase and two-phase. Transport properties like surface tension, viscosity and thermal conductivity however are not part of the equations of state and have to be supplied for with additional independent correlations.

The Helmholtz energy equation of state excludes the two-phase region. Here all the remaining properties can be calculated under the condition of mechanical and chemical equilibrium for the liquid and the vapor phase. With this assumption, all thermodynamic properties can be calculated as a function of the dew and bubble state properties and the resulting steam quality.

A comprehensive explanation of the Helmholtz energy equation of state for the refrigerant propane can be found in [7].

### 2.2 HelmholtzMedia Fluid Properties Library

The HelmholtzMedia Fluid Properties Library is an open-source Modelica library. The library is devel-

oped to be expandable and contains several different refrigerants. The library covers a wide variety of applications since the implemented EoS are valid for a wide range of state variables.

As mentioned in [6], since the EoS are implemented in their general form, the library is not optimized for speed. Also because the fluid library is designed for all conditions, it may be unstable in certain complex combination of different components, such as complex energy systems that include closed fluid circuits, which should be avoided in large scale simulations.

## 3 Methods of Calculation of EoS

### 3.1 Inverse Calculation of EoS

Common variable combinations for engineering applications often are  $(p, T)$ ,  $(p, h)$  or  $(p, s)$ . The reason for choosing these independent state variables is on the one hand an improved computational stability and on the other hand the fact that changes of state are usually expressed via these variables. There are two ways to calculate the inverse of the equations of state. The first way is to use an implicit description which was done in the HelmholtzMedia Fluid Properties Library. The second solution is to do it explicitly which is done in the new propane model discussed later on.

### 3.2 Implicit Calculation of Inverse EoS

In order to compute all other thermodynamic properties from the chosen state variables, the corresponding  $(T, \rho)$  must then be determined iteratively, since the fundamental equation of state in terms of Helmholtz energy is non-reversible.

The iteration is a root finding algorithm available in the base library of Modelica. The two-phase boundary of the fluid is described by the vapor liquid equilibrium. The dew and bubble states are then calculated by solving two out of three conservative equations: the thermal equilibrium, the mechanical equilibrium and the chemical equilibrium. Ancillary equations for the saturation pressure and the dew and bubble densities help finding these states giving start values for the iteration. They also serve for a first region check for the inverse equation of state calculation. In the close proximity to the saturation line however, the exact values must be calculated by solving the vapor-liquid-equilibrium. The vapor-liquid-equilibrium represents a state where the rate of evaporation and the rate of condensation are the same on a molecular level. It is characterized by



three conditions: thermal equilibrium, mechanical equilibrium and chemical equilibrium [8].

### 3.3 Explicit Calculation of Inverse EoS

Due to the high computational effort for the implicit calculation of state an alternative way for expressing the state variables in terms of  $(p, T)$ ,  $(p, h)$  or  $(p, s)$  is implemented in Modelica. For these calculations, polynomial fit functions were developed for a predefined region. The maximum and minimum values were chosen according to the approximate range of operation of the application. Here a temperature region between  $-10^{\circ}\text{C}$  and  $70^{\circ}\text{C}$  and a pressure region between 0.5 bars and 30 bars was chosen (see Figure 1).

For this purpose, the inverse of the whole EoS was calculated numerically. The resulted data-set was then divided into six different sets (sub cooled liquid and superheated steam for density, specific enthalpy and specific entropy). Then a surface was fitted to each data-set separately. The outcomes of this fitting were six polynomial functions of order  $5 \times 5$ .

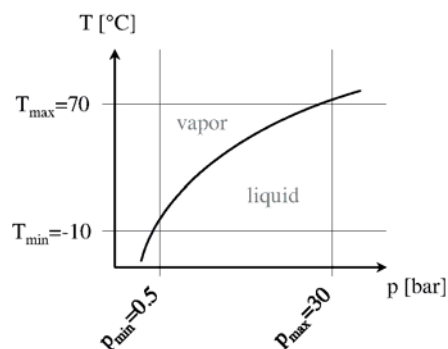


Figure 1: Range of validity for the inverse EoS

## 4 Implementation in Modelica

For all the necessary parameters, surface equations are added to the new media library. An example of such a surface is illustrated in Figure 2. The implemented media library is an extension of PartialTwoPhaseMedium which itself is an extension of PartialMedium in the Modelica Standard Library. This means that the simple propane model is also compatible with all the existing components in the Fluid library.

Moving from one region to the other makes the system not continues resulting in instabilities in solving the system of differential equations and therefore the whole simulation. To avoid these certain functions are developed for the smooth transition between each state of the fluid. These functions not

only include the correct value for the properties of propane at the discontinuous point, but also include the derivative so that smooth transitions are possible. Although this method will improve stability, since it is only applied in one point, the discontinuity of the functions can still be observed in the results.

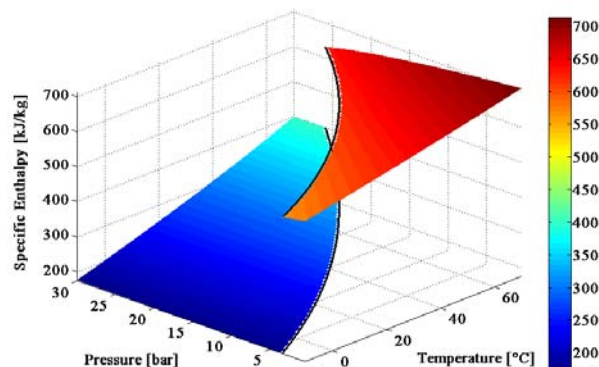


Figure 2: Illustration of the fitted surfaces in a predefined region for pressure, temperature and specific enthalpy

## 5 Results and Discussion

### 5.1 Accuracy of the Fitted Functions

As discussed earlier, it is important for the model to be accurate. For this purpose, the Sum of squared errors (CSE), coefficient of determination (RSQUARE), number of evaluated points (DFE) and root mean square error (RMSE) are calculated for the fitted curves and are shown in Table 1. The low error values indicate a very fine fitting of the data.

Table 1: Accuracy of the inverse equations

Function	SSE	RSQUARE	DFE	RMSE
$\rho(p, T)_G$	0.009132	1.000000	296656	1.7545e-4
$\rho(p, T)_L$	0.185102	1.000000	104686	0.001330
$T(p, h)_G$	0.073031	1.000000	296656	4.9616e-4
$T(p, h)_L$	2.2526e-4	1.000000	104686	4.6387e-5
$T(p, s)_G$	1.171992	1.000000	296656	0.001988
$T(p, s)_L$	7.9326e-4	1.000000	104642	8.7067e-5

For comparison between the HelmholtzMedia and the fast propane model, both fluid models are simulated at different temperature and pressures in Modelica. The relative error between the two simulations is then calculated and is shown in Table 2. The relative error in all cases is close to zero. For all the simulation parameters, this error is much smaller than the uncertainty in the system.

Table 2: Relative error between propane in HelmholtzMedia and the fast propane

$T$ K	$\rho$ mol/cm <sup>3</sup>	$p$ Rel. Err.	$C_V$ Rel. Err.	$C_P$ Rel. Err.	$w$ Rel. Err.
200.0	14.0	-2.1e-8	3.9e-8	2.9e-8	1.4e-10
300.0	12.0	-5.2e-8	1.0e-7	5.0e-8	1.3e-8
300.0	0.4	3.2e-8	8.8e-8	9.4e-9	-1.2e-8
400.0	5.0	0.0	1.9e-8	-2.0e-8	-3.8e-8
369.9	5.0	2.4e-8	6.0e-8	0.0	-3.8e-8

It can be seen that in the specified range, the relative error between two models are very close to zero. This means if the change in propane properties does not fall behind the boundaries chosen for the surface-fitting, both models can be used interchangeably with minimal compromise in accuracy.

## 5.2 Simulation Speed Comparison

Since water is the main fluid used in different energy systems, simulations with water can be set as base simulations for comparison of the simulation time for the new fluid models. Here, the simulation speed of both fluid models are compared in identical simulation models with a reference model with the simple water model available in Modelica standard library as its medium. The results are shown in Figure 3.

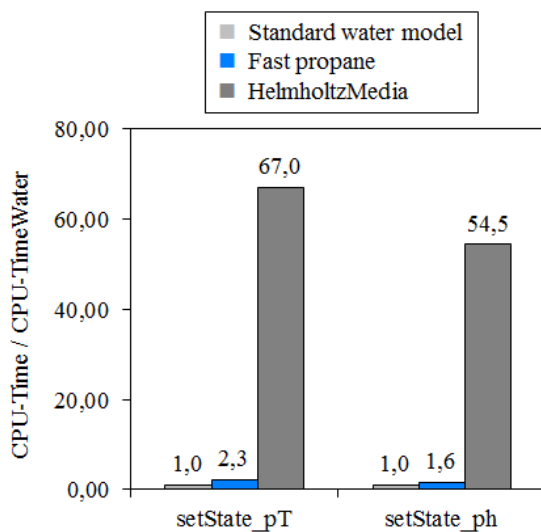


Figure 3: Simulation times for different propane models relative to water

The simulation speed of the fast propane is around 2 times slower than that of water but it is 30 times faster than the complete Propane model from the HelmholtzFluid library. This increase in simulation speed can ensure the possibility of simulating complete energy systems for a much longer period of time (monthly or yearly simulations).

## 6 Conclusions

To increase the stability and decrease the simulation time, HelmholtzMedia model for propane is modified for a certain region. The inverse calculation of the equations of states is done using several fitted surfaces instead of the actual EoS for the calculation of different variables.

It can be observed that the relative error between both models is almost zero, and on the other hand a significant increase in the simulation speed can be achieved, making the use of the fast propane model for larger simulations possible.

To conclude the differences between the two models, a simple comparison between the two models is shown in Table 3. This shows that although the HelmholtzMedia can be used for much wider range applications, for certain applications (ranges) it can be simplified so that much faster and more stable simulations are possible.

Table 3: Comparison between HelmholtzMedia and fast propane model

Property	Helmholtz Media	Fast Propane
Stability in complex systems	Not fully stable in complex energy systems with closed fluid circuits	Fully Stable between -10°C and 70°C and between 0.5 to 30 bars
Accuracy	100%	100% between -10°C and 70°C and between 0.5 to 30 bars
Simulation Speed / Simulation speed for simple water	60x	2x
Number of available media	9	1

The downside of the fast propane model is that it is only valid for a certain range of temperatures and pressures. Although this range is sufficient for many engineering applications in low temperature and low exergy heating and cooling systems, it is not sufficient for all the applications.

## References

- [1] C. Heinrich, K. Berthold “A Modelica Library for Simulation of Household Refrigeration Appliances Features and Experiences”. In: *Proceedings of the 5<sup>th</sup> International Modelica Conference*, 2006, pp. 677-684.

- 
- [2] T. Pfafferoth, G. Schmitz “Modeling and Simulation of Refrigeration Systems with the Natural Refrigerant CO<sub>2</sub>” In: *Proceedings of the International Refrigeration and Air Conditioning Conference*, 2002.
- [3] T. Pfafferoth, G. Schmitz “ Implementation of a Modelica Library for Simulation of Refrigeration Systems” In: *Proceedings of the 3<sup>rd</sup> International Modelica Conference*, 2003, pp. 197-206.
- [4] I. Bell, S. Quoilin, J. Wronski, V. Lemort “CoolProp: An Open-Source Reference-Quality Thermophysical Property Library” In: *2<sup>nd</sup> ASME ORC International Seminar*, 2013.
- [5] M. Thorade, A. Saadat “Partial Derivatives of Thermodynamic State Properties for Dynamic Simulation”. In: *Environmental Earth Sciences (online first)*, 2012. DOI: 10.1007/s12665-013-2394-z.
- [6] M. Thorade, A. Saadat “HelmholtzMedia – A Fluid Properties Library”. In: *Proceedings of the 9<sup>th</sup> International Modelica Conference*, 2012, pp. 63-69. DOI: 10.3384/ecp1207663
- [7] E. W. Lemmon, M. O. McLinden, W. Wagner “Thermodynamic Properties of Propane. III. A Reference Equation of State for Temperature from Melting Line to 650 K and Pressures up to 1000 MPa”. In: *Journal of Chemical Engineering Data* 54, 2009, pp. 3141-3180.
- [8] K. Lucas, “Thermodynamik, Die Grundgesetze der Energie- und Stoffumwandlungen”, Springer Verlag Berlin, 7. Edition, 2008. ISBN-13: 9783540686453



# Consistent Simulation Environment with FMI based Tool Chain

Edo Drenth<sup>1</sup>, Mikael Törmänen<sup>2</sup>, Krister Johansson<sup>2</sup>,  
Bengt-Arne Andersson<sup>1</sup>, Daniel Andersson<sup>1</sup>, Ivar Torstensson<sup>1</sup>, Johan Åkesson<sup>1</sup>

<sup>1</sup>Modelon AB  
Ideon Science Park, Lund, Sweden  
[info@modelon.com](mailto:info@modelon.com)

<sup>2</sup>Volvo Car Corporation  
Dept. Complete Powertrain  
Göteborg, Sweden

## Abstract

Systems engineers face the ever increasing chase for reduced time to market, while the systems to develop ever increase in complexity. Software systems design and integration processes have therefor evolved along the well-known V-cycle.

This paper will focus on the software integration for mechatronic systems as they develop fast due to high demands and challenging requirements in the automotive industry.

The development order of model in the loop (MIL), software in the loop (SIL), processor in the loop (PIL) and hardware in the loop (HIL) can be seen as state of the art practised by many systems engineers. Driver in the loop (DIL) may be in its infancy, but rapidly growing.

The novelty presented in this paper is the consistency of the plant models used in the integration chain supporting consistent model data propagation: Functional Mock-up Units (FMU) defined by the open standard of the Functional Mock-up Interface<sup>1</sup> (FMI).

*Keywords: FMI, FMU, MIL, SIL, PIL, HIL, plant models, Modelica*

## 1 Background

Volvo develops and calibrates its own engine control software. The model based design (MBD) process has been deployed for many years. The legislation on exhaust emissions and fuel consumption has become significantly stricter the past years. The change in legislation increases the burden of developing control software, calibration of parameters and validation of the mechatronic system. As a result efficiency improvements to the MBD process are required.

Experiences tell that the average development and project engineer does not feel comfortable with all aspects of MBD. It might simply be out of their comfort zone. Part of the project assignment was to bring MBD to the test and calibration engineer instead. These engineers shall be able to work with their de facto industry standard measurement, calibration and diagnostic (MCD) tools. The aim is to have a transparency for the software calibration tools as depicted in Figure 6.

## 2 Introduction

The FMI technology has been adapted fast by many modelling and simulation software vendors. This rapid adaptation of this open standard clearly is proof of an industry demand for (plant) model exchange. In the past the chain from MIL to HIL has been bridged by

many hours of manual labour to mix-and-match many, often for reasons of IP black-box, models from different sources and developed on different platforms. This was tedious work and error prone.

In the ideal case the plant model follows the entire integration process without any model modification. With the introduction of the FMI toolbox for MATLAB® Coder (FMIT-Coder) this vision is achieved.

Yet, the entire development chain from model in the loop to hardware in the loop can make use of one and the same source for a plant model exported as a Functional Mock-up Unit, FMU.

The main contributions of this paper are the use of consistent models throughout the integration workflow from desktop to test bed in the engine controller development, which by the introduction of the FMIT Coder has been made available. It is a solution to Volvo's mission to bring simulations to the test engineer. The engine test and calibration engineer can with help of industry standard protocols and tools, like the INCA<sup>9</sup> based product suite, do his/her calibration work against models or hardware.

The paper is outlined as follows. An introduction to the FMI technology is briefly drawn up, with emphasis on the toolbox available in MATLAB®. The engine and vehicle plant model based on commercially available Modelica libraries are introduced. This is to follow of a more detailed integration flow discussion with help of consistent use of FMU based plant models.

### 3 The Functional Mock-up Interface

#### 3.1 Introduction<sup>8</sup>

The FMI is a tool independent standard to support both model exchange (ME) and co-simulation (CS) of dynamic models using a combination of xml-files and compiled code.

The first version was published in 2010. The FMI development was initiated by Daimler AG with the goal to improve the exchange of simulation models between suppliers and OEMs. FMI is supported by many CAE tools and is used by automotive and non-automotive organizations throughout Europe, Asia and North America.

#### 3.2 FMI Toolbox for MATLAB

FMI Toolbox for MATLAB enables users to import FMUs into Simulink® models by means of a block-set supporting FMI for Model Exchange 1.0<sup>6</sup> and FMI for Co-simulation 1.0<sup>7</sup>. The FMU blocks can then be connected to native Simulink blocks, e.g., to support development of control systems and MIL scenarios. The FMU blocks offer a graphical user interface to parameterize the FMU, set initial conditions, configure outputs and to set the FMU log level, see Figure 1.

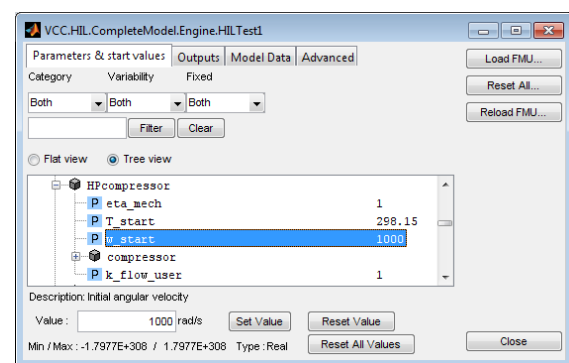


Figure 1 FMIT Dialog

#### 3.3 The FMI ME Calling Sequence

The option for co-simulation is a self-contained sampled system, but the model exchange alternative requires some tight interfacing with the master solver.

The FMI standard defines a calling sequence for simulating the FMU, see Figure 2. The FMI functions are executed in the appropriate order from the FMU block in the Simulink model. The FMU block is based upon an S-Function block which defines a list of call back functions. When the simulation loop is started, the S-function's call back functions are called which then calls the FMI functions.



For defining the engine load a vehicle model created with help of the Vehicle Dynamics Library<sup>®</sup> is deployed. The deployment of the vehicle model can either be in the same FMU as the engine model, because both used libraries are supported in Dymola, but may be two separate FMU's. The latter supports a transparent and consistent use of models all the way from desk top to an engine test bed.

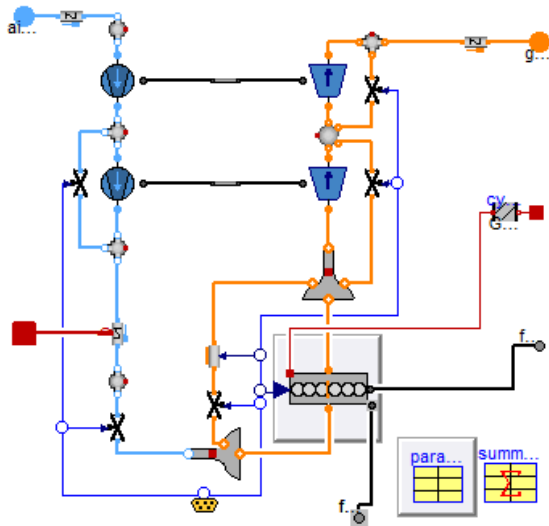


Figure 3 Engine model diagram layer

## 5 Systems integration process

### 5.1 Introduction

For completeness many integration steps are outlined and discussed below, but a deployed development process does not necessarily include all presented steps in this document.

### 5.2 Model in the loop

The accessibility and quality of code generators has improved tremendously during the nineties. This has been an enabler for development of controllers in a simulation environment.

Developing control software in a simulation environment, like Simulink<sup>®</sup> in Figure 4, allows algorithm testing in a very early stage of development. The design iterations can be much faster, because the simulation environment allows the control engineer to

quickly change algorithms and instantly simulate and thus test.

Because the plant model is supplied as an FMU and run with help of the FMI Toolbox for MATLAB<sup>5</sup> (FMIT), the development engineer that actually masters the domain of the hardware can produce a plant model in his favourite and ideal modelling environment. In this case Dymola<sup>®</sup> Engine Dynamics Library<sup>®</sup>.

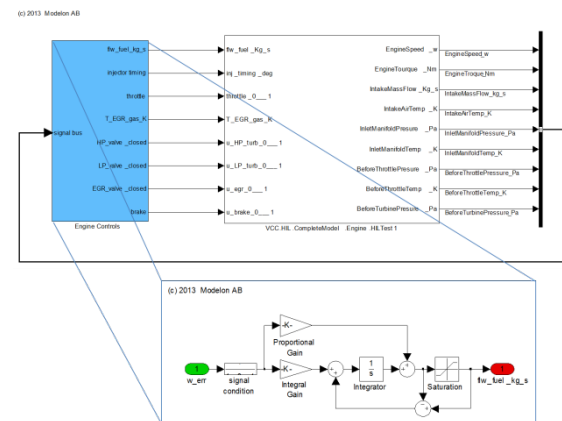


Figure 4 A MIL (blue block) example of a simple engine controller and an EDL based engine model exported as an FMU (white block)

### 5.3 Software in the loop

Once the algorithms in the MIL stage perform as designed the controller model can be transformed into c-code with a coder like Simulink Coder or TargetLink<sup>®</sup> including debug information. This exported software code can in its turn be linked with external sources of code or manual written code if desired.

The created code can firstly be tested in Simulink as shown in Figure 5. This solution mimics the MIL solution very closely.

The created code can also be hooked up to communicate with other systems and sub-systems in for instance Silver<sup>™</sup> by QTronic<sup>3</sup>. Silver supports virtual module integration and test automation. An important benefit for Volvo using Silver is the support it has for the already deployed engine calibration tools and protocols. The calibration engineer won't see the difference if s/he is calibrating a virtual or a real engine.





of using virtual vehicle models is that the environmental conditions are more controllable than in real vehicles tests.

The consistent usage of plant models across the different experimental domains allows propagation of parameters for the physical models and data for the empirical models for the plant models at different fidelity level all the way back to MIL.

(CSW) is interacting with the physical plant models.

Simulink Coder or TargetLink allow the CSW to be exported to Silver. This solution allows for SIL and VPIL. Silver supports standard CAN protocols and thus standard already available measurement, calibration and diagnostic (MCD) software can be used.

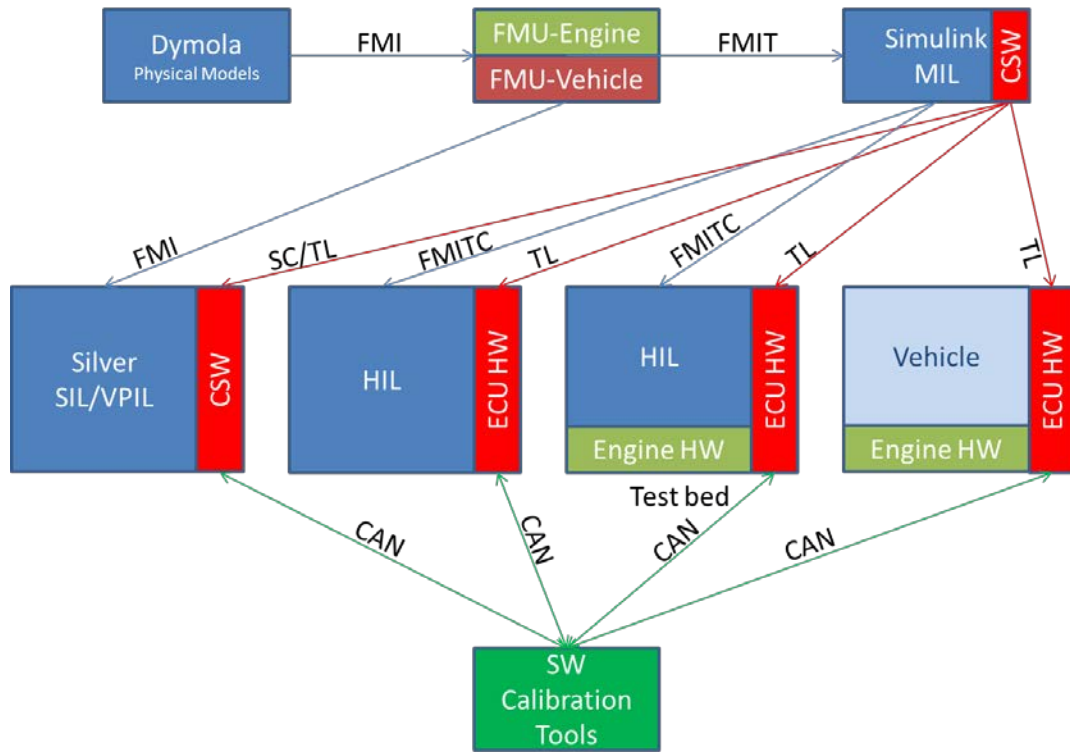


Figure 6 FMU’s central role in control system integration process. For sake of simplicity the PIL solutions are omitted. TL=TargetLink, SC=Simulink Coder, CSW=Control Software.

Data retrieved from tests will be part of data- and model regression and the process is thus improving quality of results over time.

### 6 Process summary

The process summary is depicted in Figure 6. Dymola is the environment where the plant models for the physical systems, engine and vehicle in this case, are created. These models are exported as FMU’s. On binary level these FMUs can with help of FMIT be used in Simulink and can also be natively imported to Silver. In Simulink the control software model

The next logical step is the export of plant models to the HIL environment with FMIT Coder. The model fidelity might be lower to account for the real time constraints this solution has. The CSW is in the real ECU hardware exported with TargetLink. Of course the standard MCD software can be invoked to alter the ECU calibration.

Currently the final stage will be the engine test bed that has its loads determined with an FMU for the vehicle with driveline. For the MCD software this is exactly the same use case as the HIL solution.

The test bed results can be fed back into the model parameters and data to evolve fidelity and quality of models. An iterative process is created to continuously improve model fidelity and quality.

At the end of the process of course, the engine and engine controllers are assembled in the vehicle. The same MCD-toolset can be used.

## 7 Conclusions

With the speed FMI technology is embraced in the industry is clear sign it has solved a long existing challenge for systems integration and validation engineers. Often tedious and error prone manual modifications to adopt the supplied models and data to different simulation environments have become a technology of the past with the introduction of FMI compliant FMUs.

With the FMIT-Coder the FMI based chain of FMU deployment is complete. Yet the entire suite of validation and verification development stages is covered by FMI technology.

The consistent use of FMI compliant models has been an enabler for improved work flow efficiency and model quality of the MBD process.

Part of the project assignment was to bring MBD to the test and calibration engineer. These engineers shall be able to work with their de facto industry standard measurement, calibration and diagnostic tools. The aim to have a transparency for the MCD tools is accomplished and depicted in Figure 6.

## 8 Copyright notice

All trademarks mentioned belong to their respective owners.

## 9 References

1. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.-V., Wolf, S., *The Functional Mockup Interface for Tool independent Exchange of Simulation Models*, 8<sup>th</sup> Modelica Conference, Dresden, Germany, 2011
2. Liebezeit, Bräuer, Serway, Junghanns: *Virtual ECUs for developing automotive transmission software*, 10th CTI Symposium Innovative Fahrzeug-Getriebe Hybrid- und Elektro-Antriebe, 5.-8.12.2011, Berlin, Germany
3. Junghanns, A., *Virtual integration of Automotive Hard- and Software with Silver*, Qtronic GmbH, Berlin
4. Andreasson, J., Andersson, D., Batteh, J., Gohl, J., Griffin, J., Krueger, I., *Integrated simulation of a e4WD vehicle using Modelica*, Advances in Automotive Control, Volume#7, Part#1, 2013
5. *FMI Toolbox User's Guide 1.7*, Modelon AB, Lund, Sweden, 2013
6. *Functional Mock-up Interface for Model Exchange, Version 1.0*
7. *Functional Mock-up Interface for Co-Simulation, Version 1.0*
8. <https://www.fmi-standard.org/>
9. *Measurement, ECU Calibration, and Diagnostics –Development Solutions for Automotive Embedded Systems*, ETAS GmbH, Stuttgart, Germany, 2010



# A MATLAB to Modelica Translator

Mohammad Jahanzeb<sup>1</sup>, Arunkumar Palanisamy<sup>1</sup>, Martin Sjölund<sup>1</sup>, Peter Fritzson<sup>1</sup>

<sup>1</sup>PELAB – Programming Environment Laboratory

<sup>1</sup>Department of Computer and Information Science

Linköping University, SE-581 83 Linköping, Sweden

mjahanzeb@live.com, {arunkumar.palanisamy, martin.sjолund, peter.fritzson}@liu.se

## Abstract

Matlab is a proprietary, interactive, dynamically-typed language for technical computing. It is widely used for prototyping algorithms and applications of scientific computations. Since it is a dynamically typed language, the execution of programs has to be analyzed and interpreted which results in lower computational performance. In order to increase the performance and integrate with Modelica applications it is useful to be able to translate Matlab programs to statically typed Modelica programs.

This paper presents the design and implementation of Matlab to Modelica translator. The Lexical and Syntax analysis is done with the help of the OMCCp (OpenModelica Compiler Compiler parser generator) tool which generates the Matlab AST, which is later used by the translator for generating readable and reusable Modelica code.

*Keywords: Modelica, MetaModelica, Matlab, OMCCp, translation.*

## 1 Introduction

The Matlab language is dynamically typed; it does not have explicit type declarations. A variable's type is implicit from the semantics of its operations and the type is allowed to dynamically change at runtime.

These features improves ease of use for prototyping and interactive use, but add heavy run-time overheads, such as runtime type checking, array bounds checking and dynamic resizing, to its interpretive execution. Therefore, Matlab programs often run slower than their counterparts which are written in conventional statically typed programming languages.

The main goal of this work is the development of a translator that accepts Matlab programs as input and

generates Modelica code as output which is suitable for static compilation. Due to the complexity of the Matlab language, a realistic goal is to develop a translator for a subset of Matlab.

The translation task of Matlab to Modelica code mainly involves the front-end implementation of the Matlab to Modelica compiler. The OMCC (OpenModelica Compiler Compiler) compiler generation tool, which has been developed as a part of the OpenModelica project, can be used as a parser and translator generator extended with advanced error handling facilities. The tool is implemented in MetaModelica and integrated with the MetaModelica semantics specification language based on operational semantics for generating executable compiler and interpreter modules.

The OMCCp part of the OMCC tool makes the implementation of the first two stages of a compiler much easier and saves time. We have to just write the lexer and parser rules for the Matlab language and input them to OMCCp to generate the appropriate lexer and parser modules in MetaModelica. The generated parser builds the Abstract Syntax Tree (AST) for the Matlab source code that is parsed.

The Matlab AST is later used by the second phase of the Matlab-to-Modelica translator which performs a series of internal transformations and finally generates the Modelica-AST which is unparsed to readable Modelica code.

There exists a large body of computational algorithms which are programmed in the Matlab language. The need to have an easy way of using and incorporating such algorithms within Modelica models, as well as achieving improved performance, motivated the development of the Matlab to Modelica translator in the OpenModelica project.

This paper is structured as follows: Section 2 presents some related approaches. Section 3 describes the different steps involved in generating a compiler from

specifications in different specification formalism. Section 4 explains the design and implementation of the Matlab to Modelica translator with main focus on the translator. The translator description is for the subset of the Matlab language grammar for which translation to Modelica code is supported. Section 5 explains the type inference approaches. Section 6 describes the Modelica unparser. Section 7 presents Test results and performance measurements. Finally, Section 8 concludes the paper with a short discussion of achieved results.

## 2 Related Work

In this section we discuss a few related approaches for the compilation of Matlab code to statically typed languages.

The Matlab tamer is an extensible object-oriented framework for generation of static programs from dynamic Matlab programs implemented in Java. The Matlab Tamer supports a large subset of Matlab. It builds a complete call graph, transforms every function into a reduced intermediate representation, and provides typing information to aid the generation of static code [15]. In an earlier student project we tried to use the Matlab tamer framework directly for a translator to Modelica. However, that project was not completed.

MCFOR: A Matlab to Fortran95 compiler is designed for translating Matlab code to Fortran95. It generates readable and reusable Fortran95 code. MCFOR had very limited support for built-in functions. It showed that the numerical and matrix features of Fortran95 are a good match for the compiled Matlab, and that the static nature of the language, together with powerful Fortran95 compilers provides the potential for high performance [16].

## 3 Background

### 3.1 Generating Compiler Phases

The different phases of the compiler can be generated from a formal specification in different formalisms, as depicted in Figure 1.

Generally a compiler is divided into two parts, the front-end and the back-end. The scanner and parser constitute the front end phase whereas the optimization and code generator constitute the back-end phase of the compiler. In this paper we focus on the front-end parts of the compiler [1] [2], and use the existing Modelica unparser to generate the Modelica code from the internal Modelica AST

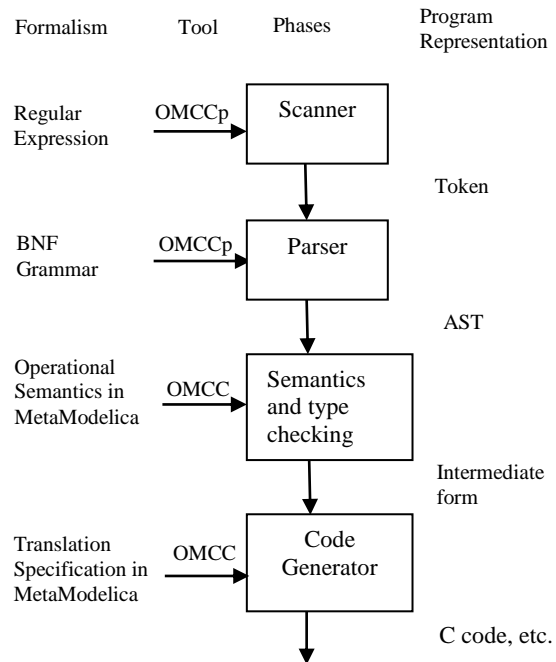


Figure 1. Structure of Compiler Generation using OMCC

### 3.2 Lexical Analysis

The Lexical analysis, performed by a scanner, is the first stage of the compilation process. It receives the source code as input and generates tokens. It also identifies the special tokens defined by the specified language making it simpler for the next phase of the compiler.

The structure of the tokens are usually specified by the use of regular expressions. There are several tools available that automate the labor of constructing the transition rules to identify the tokens for a scanner. We use the Flex tool for this purpose which generates C code; the generated C code containing tables is used by OMCCp to generate the appropriate lexer components in MetaModelica [1] [2].

### 3.3 Syntax Analysis

The syntax analysis, also called parsing, is the second stage of the compilation process. The parser takes the tokens generated by the lexer and determines whether the tokens are constructed according to the rules of the grammar. By doing this it creates the Abstract Syntax Tree (AST) if the input conforms to the defined grammar and otherwise reports an error message.

The AST is used as input to the back-end. The back-end uses the AST for type checking, optimization, and finally generates machine specific code. The grammar rules are usually specified in the form of BNF (Backus Normal Form). We use the BNF grammar rule format of the popular Bison tool for writing the grammar rules;

the generated C code contain parse tables that are used by OMCCp to generate appropriate parser components in MetaModelica [1][2].

### 3.4 Semantic Analysis

The semantic analysis is the phase in which the compiler adds semantic information to the parse tree and adds semantic information to the symbol table. This phase also performs semantic checks such as type checking (checking for type errors), or object binding (associating variable and function references with their definitions), or definite assignment (requiring all local variables to be initialized before use), rejecting incorrect programs or issuing warnings.

Semantic analysis usually requires a complete abstract syntax tree, meaning that this phase logically follows the parsing phase, and logically precedes the code generation phase, though it is often possible to fold multiple phases into one pass over the code in a compiler implementation.

The translator which takes the Matlab AST as input performs a series of internal transformation to produce the corresponding translated Modelica AST which can be unparsed to the generated textual Modelica code [1] [2].

## 4 Design

The design architecture of the Matlab to Modelica translator is depicted in Figure2.

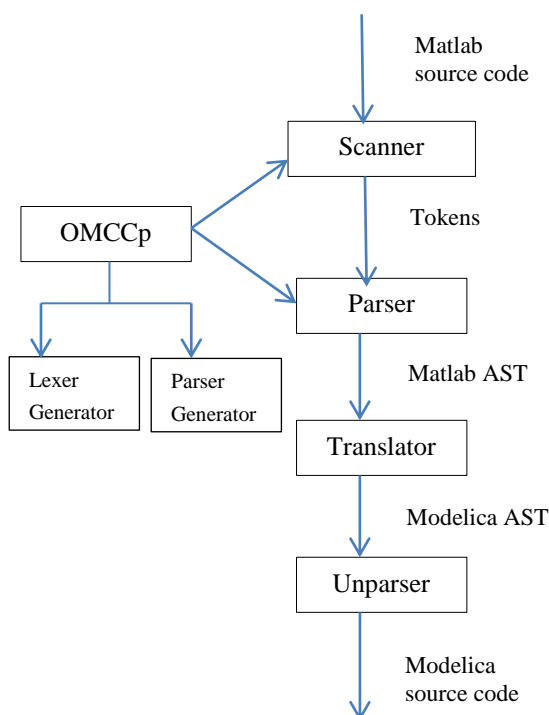


Figure 2. Structure of Matlab to Modelica translator

The translator contains a scanner and a parser for the Matlab source code, which build the Matlab abstract syntax tree during parsing. The scanner and the parser are implemented using the OMCCp tool which generates appropriate lexer and parser components in MetaModelica. The result of the parser is used as input to the translator which performs a series of internal transformations and generates a Modelica AST. The last stage is the Unparser where the Modelica AST is unparsed to generate the Modelica source code. The scanner, parser and the translator are implemented in MetaModelica. In this paper we focus on the Matlab to Modelica translator; for detailed information about OMCCp see [1] [2].

### 4.1 LexerGenerator

The LexerGenerator is the main package of the lexer which generates the necessary lexer subsystems in MetaModelica. This module generates three packages in MetaModelica namely LexerModelica.mo, LextableModelica.mo and LexcodeModelica.mo. For more information about these packages see [1] [2].

#### 4.1.1 Lexer.mo

Lexer.mo is the main file which contains the calls to other functions in Lextable.mo and LexCode.mo that constitute the lexer. The main function of this package is to load the source code file and recognize all the tokens described by the grammar.

For recognizing a token the lexer.mo runs DFA (Deterministic Finite Automata) based on the transition arrays found in Lextable.mo. When it reaches an acceptance state it calls the function `action` in LexCode.mo which returns list of tokens that are input to the parser. The interface of the function which does this operation is given below.

```

function scan
  input String fileName "input source code file";
  input list<Integer> program "source code as a stream of Integers";
  input Boolean debug "flag to activate the debug mode";
  output list<OMCCTypes.Token> tokens
  "return list of tokens";
end scan
  
```

### 4.2 Parser Generator

The ParserGenerator package is the main package of OMCCp that generates the parser which performs the syntax analysis of the compiler. ParserGenerator generates four packages in MetaModelica namely TokenModelica.mo, ParserTable.mo, ParserCodeModeli

ca.mo and ParserModelica.mo. For more information about these packages see [1] [2].

#### 4.2.1 Parser.mo

The main function of this package is to efficiently convert the list of tokens received by the Lexer into Abstract Syntax Tree (AST). The package also contains the implementation of the LALR algorithm. For performing this task the package uses the parse table located in the package ParseTable.mo, to perform the shift-reduce action calls it uses the package ParserCodeModelica.mo. The interface of the function which starts the construction of AST is given below.

```
function parse "realize the syntax
  analysis over the list of tokens and
  generates the AST tree"
input list<OMCCTypes.Token> tokens "list
  of tokens from the lexer";
input String fileName "file name of the
  source code";
input Boolean debug "flag to output
  debug messages that explain the states
  of the machine while parsing";
output Boolean result "result of the
  parsing";
output ParseCode.AstTree ast "AST tree
  that is returned when the result
  output is true";
end parse;
```

### 4.3 Translator

The transformation *translator* is the third stage of the implementation which takes the Matlab AST as the input and performs a series of internal transformation and finally generates the Modelica AST. The translator package contains several functions which perform the above tasks by finding a possible mapping to a Modelica AST data structure. The implemented translator supports a particular subset of the Matlab language which is discussed in more detail in the following section.

### 4.4 Primary Function

A Matlab program consists of a list of files called M-files, which include a sequence of Matlab commands and statements. We can split the syntax of a Matlab function into three parts. First is the declaration part, which consists of the function name, the input and output formal parameters. The next part is the function body which consists of a list of statements which ends with the keyword `end`. A sample function in Matlab is given below.

```
function perfect = isperfect(value)
  sum = 0;
  for (divisor = 1 : value - 1)
    result = value / divisor;
```

```
    if (result == floor(result))
      sum = sum + divisor;
    end
  end
  if (sum == value)
    perfect = 1;
  else
    perfect = 0;
  end
end
```

The function name is `isperfect` which is defined in the declaration section. The input and output parameters are `value` and `perfect` respectively.

The body of function starts right after the declaration line and ends with keyword `end` which also ends the function. Due to the dynamic nature of the language there are no data type declarations in the Matlab code. All variables inside the function body are local to the `isperfect` function.

#### 4.4.1 Translation of Matlab to Modelica function

A Modelica function consists of three parts. The first part is similar to a Matlab function which contains function name, input, and output formal parameters. Since the Modelica language has static typing we have to define each function formal parameter with its proper data type as well as `input` and `output` keywords.

The next part is `protected`. All local variables apart from `input` and `output` formal parameters have to be declared in the `protected` section with proper data types. Finally the body of function starts with the keyword `algorithm` and ends with `end` keyword. An example of the Matlab function `isperfect` translated to Modelica is presented below.

```
function isperfect
  input Real value;
  output Real perfect;
protected
  Integer sum;
  Real result;
algorithm
  sum:=0;
  for divisor in 1:value - 1 loop
    result:=value / divisor;
    if result == floor(result) then
      sum:=sum + divisor;
    end if;
  end for;
  if sum == value then
    perfect:=1;
  else
    perfect:=0;
  end if;
end isperfect;
```



#### 4.5 Translation of function declaration statements

In the function declaration the translator assigns proper data types to the input and output formal parameters. The translator determines the data types of formal parameters once the whole function body has been traversed. An example of the translation is presented below.

##### Matlab

```
function perfect = isperfect(value)
```

##### Modelica

```
function isperfect
  input Real value;
  output Real perfect;
```

#### 4.6 Identification and Translation of Local identifiers

All variables defined inside the Matlab function body are local variables. Modelica declares all local variables in the translated function under the protected section. Therefore all those local variables have to be assigned proper data types before declaration in the protected section.

For translation of this phase we have to identify all local variables first since the function body contains input and output formal parameters and Modelica does not support re-declaration of variables. The translator then assigns the proper data types. The process of identifying the local variables is depicted in Figure 3.

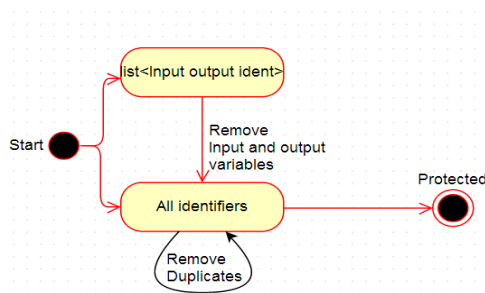


Figure 3. Declaration of variables

To perform this task we declare two lists of strings. Assume that there are two lists: a `param` and an `ident` list. The former contains input and output parameters and the latter contains all variables of the function including input and output parameters.

The translator compares both lists and removes all those variables from the `ident` list which were found in the `param` list. Now the `ident` list only contains local variables left but it might have duplicate variables.

To remove the duplicates compare the list with itself and remove all duplicates.

#### 4.7 Identification and Translation of constant variables

Matlab does not support constant declaration of variables as in Modelica with the prefix `constant` keyword. Here we mean identifiers whose right hand side is equal to any constant value, i.e., real, integer, logical or array not any variable. For example, `sum = 10` where 10 is an integer scalar. All these variables are declared under the `protected` heading with their relevant data types. Take a look at the following translation:

##### Matlab

```
function [sum] = add_scl_mat(scl1)
  real1 = 10.5;
  mat2 = [1,2,3;4,5,6];
  sum = mat2 + scl1 + real1;
end
```

##### Modelica

```
function add_scl_mat
  input Real scl1;
  output Real sum[:, :];
  protected
    Real real1;
    Integer mat2[:, :] = [1,2,3;4,5,6];
  algorithm
    real1:=10.5;
    sum:=mat2 .+ scl1 .+ real1;
  end add_scl_mat;
```

The translator identifies two local variables. First the variable `real1` whose right hand side is a real number, i.e., 10.5. The next one is `mat2` whose right hand side is a two dimensional integer array, i.e., [1,2,3;4,5,6].

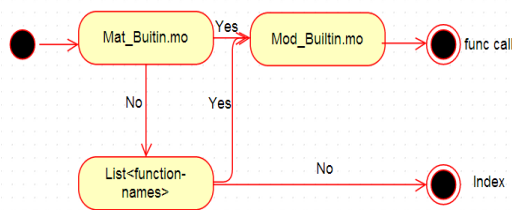
The translator identifies their respective data types from the right hand side value and then performs translation. The translated scalar identifiers only have type declarations in the protected section, the assignment of values is under the algorithm section whereas arrays have both declaration and assignment in the protected section.

#### 4.8 Translation of Function Body

The function body of the Matlab function is translated into a Modelica algorithm section. The body is composed of a list of statements where every statement is translated to the relevant Modelica statement.

#### 4.8.1 Identification and Translation of Function Call

The syntax for function call and array index operation is the same in Matlab using round parentheses, i.e. `sqrt(i)`. Thus `sqrt` can be a function call or an index operation. Matlab decides this during execution time. However, Modelica has different syntax for index operations. It uses brackets `[]` instead of parentheses `()`. Thus, `sqrt[i]` should be used if `sqrt` is an array that should be indexed. The function call has the same syntax as in Matlab, i.e., `sqrt(i)`.



**Figure 4.** Process of Identifying function call

Figure 4 presents the process where a function call is translated. For correct translation the translator must differentiate between function call and index operation.

The function call in the original Matlab code can either be a built-in function in Matlab, a sub-function, or a recursive function. For proper generation of translated code the translator copies the names of main functions and sub-functions into a list of strings.

Secondly, the mostly used Matlab built-in functions are collected and placed in the file `Mat_Builtin.mo`. This file works as a dictionary. If the translator finds any function call in input Matlab code it compares the name of the function in both the dictionary and the list of collected function names.

If there is a match in any list then the translator looks for a similar function in the other file `Mod_Builtin.mo` where Matlab and Modelica functions were placed which have a similar purpose but have different names.

For example, the `disp('text here')` function is used in Matlab for print/display purpose is replaced to a `print("text here")` function in OpenModelica because it has a similar purpose. Similarly `rdivide()` to `div()`, `diag()` to `diagonal()` etc. If the translator does not find it, it interprets the identifier as an array that should be indexed and the translator replaces parenthesis `()` by brackets `[]`.

#### 4.9 Identification and Translation of Anonymous function to sub-function

An anonymous function is just like a standard Matlab function. It is placed inside the body of a function and accepts inputs and return output as standard Matlab functions. However it contains only a single executable statement.

An example of a sub-function translation is presented below.

##### Matlab

```
function [result] = fnc(x)
    y = 2;
    sqrt = @(x) x^y;
    result = sqrt(x);
end
```

##### end

##### Modelica

```
function fnc //primary function
    input Real x;
    output Real result;
    protected
        Real y;
    algorithm
        y:=2;
        result:=sqrt(x, y);
    end fnc;

function sqrt //sub-function
    input Real x;
    input Real y;
    output Real sqrt;
    algorithm
        sqrt:=x ^ y;
    end sqrt;
```

Modelica does not support such functions. Therefore we handle them differently. The Translator converts an anonymous function to a sub-function in Modelica. The translator traverses the whole AST to find out whether the body of a function contains any anonymous function or not. If yes, then translator copies the whole anonymous expression from the original code and removes that node from the AST. Afterwards it converts it into a sub-function in Modelica. In a function call the translator replaces the input parameters with sub-function input formal parameters. As we see in the code `sqrt(x)` has only one formal parameter but in translated Modelica code we find two parameters, `x` and `y`.

#### 4.10 Translation of looping statements

Both languages use similar syntax in for- and while-loop header expressions. The translator only translates the `=` sign to the `in` keyword and the postfix `loop` keyword in an header expression.

Also, the square brackets `[]` replaces the braces or curly brackets `{}`. Matlab ends a for loop with an `end`

end for keywords. Therefore translator adds the end for keywords at the end. The same ending procedure is followed for while loops and if statements as end while, end if. An example of the translation is presented below.

#### .Matlab

```
for m = 1:14
end
```

#### Modelica

```
for m in 1:14 loop
end for;
```

#### Matlab

```
while nFactorial < 100
end
```

#### Modelica

```
while nFactorial < 100 loop
end while;
```

### 4.11 Translation of if statements

The if statement is also similar in both languages. The translator adds the then keyword in the headers of if expressions and the endif keyword to end the expressions. An example of the translation is presented below.

#### Matlab

```
if(result == floor(result))
    sum = sum + divisor;
end
```

#### Modelica

```
if result == floor(result) then
    sum:=sum + divisor;
end if;
```

### 4.12 Translation of Switch Statement

Modelica does not currently support the switch statement. When the translator finds a switch statement in the original Matlab code it translates it into if-else statements with possible elseif branches. An example of the translation is presented below.

#### Matlab

```
switch mynumber
case -1
    disp('negative one');
case 0
    disp('zero');
case 1
    disp('positive one');
otherwise
    disp('other value');
end
```

#### Modelica

```
if mynumber == -1 then
    print("negative one");
elseif mynumber == 0 then
    print("zero");
elseif mynumber == 1 then
    print("positive one");
end if;
```

### 4.13 Sub-function

A Matlab file can contain more than one function. If it contains multiple functions, then the first function is the primary function which can be accessed in other M files and the rest of the functions are secondary functions which are called as sub-functions.

A sub-function is only accessed by its primary function. All sub-functions are translated into sub-functions in Modelica and this process follows the same procedure for translation as for the main/primary function.

## 5 Type Inference

We have adopted the MCFOR "A Matlab To Fortran95 Compiler" approach for evaluating the types in our translator. It starts by analyzing each assignment statement on the right hand side (RHS) of the expression and assign types [16].

### 5.1 Statement where Right Hand Side is a constant

Constant values are very precise sources for assessing data types. They are the starting points for our type inference process. If the translator finds an assignment statement whose right hand side is a constant (e.g.) temp = 10, in Modelica such an expressions will be declared as an integer data type.

#### Example

```
temp = 10 => Integer temp
temp = 10.5 => Real temp
temp = [1,2,3,4] => Integer temp[1,4]
"size is 1x4"
temp = [1,2;3,4] => Integer temp[2,2]
"size is 2x2"
```

### 5.2 Statement where Right Hand Side is Built-in function

Built-in functions in Matlab are well defined therefore these functions also provide enough information for assessing data types. If an assignment statement contains built-in function on right hand side translator will easily assess its type from database where we have a list of all built-in functions supported by Modelica with their return type. (e.g.) zeros(n,m) is an n-by-m ma-

trix of zeros. If  $n$  and  $m$  is equal to 2 then translator will declare left hand side identifier as a two dimensional Integer array.

Example

```
temp = zeros(2,2) => Integer temp[2,2]
"size is 2x2"
temp = ones(1,3) => Integer temp[1,3]
"size is 1x3"
```

### 5.3 Statement where Right Hand Side is a computational expression and contains relational operator

Relational operators always return the results of the logical type. (e.g.)  $c = a < b$ .

### 5.4 Statement where Right Hand Side is a computational expression and contain arithmetic operator (array)

Matlab operators not only provide the shape and size information for the result but also provide constraints on the type and shape of operand. The assignment statement where right hand side is matrix computational expression,  $c = a*b$  where  $a$  &  $b$  are matrices.

In order to evaluate such an expression in Matlab the inner dimensions must agree. If  $a$  &  $b$  have a shape of  $n$ -by- $m$  and  $p$ -by- $q$ , then  $m$  must be equal to  $p$ . The generated result will have the shape of  $n$ -by- $q$ . For example if we have an expression  $temp = a * b$  where dimension size of  $a$  is  $1 \times 3$  and  $b$  is  $3 \times 3$  then  $temp$  have dimension  $1 \times 3$  and translator declares as an Integer  $temp[1,3]$ .

### 5.5 Statement where Right Hand Side is a computational expression and contain arithmetic operator (scalar)

Consider the following expression  $temp = a / b$  where  $a$  &  $b$  are Integer scalar variables. To avoid data loss the translator promotes the left hand side variable as real data type.

### 5.6 RHS is not equal to LHS

In Matlab we can re-create any variable with different type assignment and every assignment can create a type conflict where left hand side variable is different from the type of the right hand side expression. In such cases the translator performs typecasting on right hand side expression and defines it with proper data type.

## 6 Unparser

When translating the code to an abstract syntax tree in Modelica we need to unparse the AST to get a readable output, i.e., Modelica source code. The unparsing is performed by the package Dump which is already present in the OpenModelica Compiler. The interface of the function which starts the unparsing is given below.

```
public function unparseStr
  "Pretty prints the Program, i.e. the
   whole AST, to a string."
  input Absyn.Program inProgram;
  input Boolean markup
  output String outString;
end unparseStr;
```

## 7 Testing

We tested a number of Matlab programs to demonstrate that the translator was working properly.

A sample Matlab function to calculate the area is presented below

```
function area_sum = area_inside(radius,
num_boxes)
  box_length = (2.0*radius)/num_boxes;
  box_rad = box_length*0.5;
  box_area = box_length*box_length;

  area_sum = 0;
  for (xi = 1 : num_boxes)
    xc = - 1 + box_rad + box_length*(xi -
      1);
    for (yi = 1 : num_boxes)
      yc = 1 + box_rad + box_length*(yi -
        1);

      dist = sqrt((xc*xc) + (yc*yc));
      if (dist < radius)
        area_sum = area_sum + box_area;
      end
    end
  end
end
```

The above input is translated to the following Modelica source code which gives the same result..

```
function area_inside
  input Real radius;
  input Real num_boxes;
  output Real area_sum;
protected
  Real box_length;
  Real box_rad;
  Real box_area;
  Real xc;
  Real yc;
  Real dist;
algorithm
  box_length:=2.0 * radius / num_boxes;
  box_rad:=box_length * 0.5;
  box_area:=box_length * box_length;
  area_sum:=0;
```

```

for xi in 1:num_boxes loop
  xc:=-1 + box_rad + box_length *
    (xi - 1);
  for yi in 1:num_boxes loop
    yc:=1 + box_rad + box_length *
      (yi - 1);
    dist:=sqrt((xc * xc) + (yc * yc));
    if dist < radius then
      area_sum:=area_sum + box_area;
    end if;
  end for;
end for;
end area_inside;

```

## 7.1 Performance Evaluation

In this section we present the performance of the Modelica code generated by the translator which is compared to the execution time with the corresponding Matlab code in Matlab. Before the comparison we made sure that both codes generate same results. The measurements are listed in Table 1 below.

**Table 1.** Time measurement of Matlab with Modelica

Model	Scalar/Array	Matlab(ms)	Modelica(ms)
Closure	2D-array	185.90	26.49
IsPerfect	scalar	198.60	11.65
Finite	2D-array	69.98	55.25
AreaInside	scalar	180.05	16.72
MySort	1D-array	218.45	26.49
SwapVector	1D-array	80.67	18.02
BubbleSort	1D-array	109.34	81.96

From Table 1 we can clearly see that the generated Modelica code has better performance than the Matlab code.

## 8 Conclusion

In this paper we have presented a Matlab to Modelica translator that works for a specific subset of Matlab. The translator generates readable and reusable Modelica code. It can handle the following Matlab constructs:

- *Functions*: Simple, Nested and Sub-functions.
- *Loops*: For, Nested For, While, Nested While, and break.
- *Statements*: If, If else, If elseif else and Switch.
- *Program Termination*: return.
- *Arithmetic Operators*: Plus +, Unary plus ++, Minus -, Unary minus --, Matrix multiply \*, Array multiply .\*, Matrix power ^, Array power .^, Backslash or

left matrix divide \, Slash or right matrix divide /, Left array divide .\, and Right array divide ./.

- *Relational Operators*: Equal ==, Not equal ~=, Less than <, Greater than >, Less than or equal <= and greater than or equal >=.
- *Logical Operators*: Element-wise logical AND &, Element-wise logical OR | and Logical NOT ~.

The generated code can then be statically compiled by a Modelica compiler which usually result in better runtime performance.

## 9 Acknowledgments

This work is done within the OpenModelica project. The OpenModelica work is supported by the Open Source Modelica Consortium.

## References

- [1] Edgar Alonso Lopez-Rojas. OMCCp: A Meta-Modelica based parser generator applied to Modelica. Master's-Thesis, Linköping University, Department of Computer and Information Science, PELAB- Programming Environment Laboratory, ISRN:LIU-IDA/LITH-EX-A--11/019--SE, May 2011.
- [2] Arunkumar Palanisamy. Extended MetaModelica based Integrated Compiler generator. Master's-Thesis, Linköping University, Department of Computer and Information Science, PELAB-Programming Environment Laboratory, ISRN:LIU-IDA/LITH-EX-A--12/058--SE, October 2012.
- [3] Peter Fritzson. Principles of Object-oriented modeling and simulation with *Modelica 2.1*. Wiley-IEEE Press, 2004.
- [4] Peter Fritzson, Adrian Pop and Martin Sjölund. Towards Modelica4 Meta-Programming and Language Modeling with MetaModelica 2.0, Technical reports Computer and Information Science Linköping University Electronic Press, ISSN:1654-7233; 2011:10
- [5] Peter Fritzson and Adrian Pop. Meta-Programming and Language Modeling with MetaModelica 1.0. Technical reports Computer and Information Science Linköping University Electronic Press, ISSN: 1654-7233. 2011:9.
- [6] Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman. *Compilers Principles, Techniques, and Tools*, Second Edition. Addison-Wesley, 2006.
- [7] Peter Fritzson et al. Compiler Construction laboratory assignments. Compendium, Bokakademien,

- Linköping University, Department of Computer and Information Science, 2011.
- [8] Michael Burke and G.A. Fisher Jr. A practical method for syntactic diagnosis and recovery. In *Proceedings of the 1982 SIGPLAN symposium on Compiler Construction*, 1982.
- [9] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.6*, November 2010. <http://www.openmodelica.org>
- [10] TheMathWorks,Inc.MATLAB <http://www.mathworks.se/>
- [11] Luiz De Rose and David Padua. Techniques for the Translation of MATLAB Programs into Fortran 90, *ACM Transactions on Programming Languages and Systems*, Vol 21, March 1999.
- [12] Luiz De Rose and David Padua. *A MATLAB to Fortran 90 Translator and its Effectiveness*. ISBN:0-89791-803-7, 1996.
- [13] Pramod G. Joisha, Abhay Kanhere, Prithviraj Banerjee, U. Nagaraj Shenoy, Alok Choudhary. The Design and Implementation of a Parser and Scanner for the MATLAB Language in the MATCH Compiler, 1999.
- [14] Pramod G. Joisha, U. Nagaraj Shenoy, Prithviraj Banerjee. *An Approach to Array Shape Determination in MATLAB*. Technical report no. CPDC-TR-2000-10-010, Oct 2010.
- [15] Anton Dubrau and Laurie Hendren. *Taming Matlab*, Sable Technical Report No. sable-2011-04. McGill University, School of Computer Science, April, 2011.
- [16] Jun Li. *MCFOR: A MATLAB TO FORTRAN 95 COMPILER*. McGill University, School of Computer Science, August, 2009.

## 10 Appendix

The Appendix presented below represents the AST data structures used for the construction of MATLAB source code.

```
Absynmat.mo
```

```
encapsulated package AbsynMat
public type Ident = String;
uniontype Start
  record START
    User_Function usr_fun;
    Separator sep;
    list<Statement> stmt_lst;
  end START;
end Start;

uniontype User_Function
  // Begin defining a function.
  record START_FUNCTION
    Ident fname;
```

```
list<Parameter> prm;
Option<Separator> sep;
list<Statement> stmt_lst;
Statement stmt_2nd;
end START_FUNCTION;

// Finish defining a function.
record FINISH_FUNCTION
  list<Decl_Elt> ret;
  User_Function usr;
end FINISH_FUNCTION;
end User_Function;

uniontype Argument
  record ARGUMENT
    Expression exp;
  end ARGUMENT;

  record VALIDATE_MATRIX_ROW
    Argument arg_lst;
  end VALIDATE_MATRIX_ROW;
end Argument;

uniontype Command
  record TRY_CATCH_COMMAND
    Separator sep;
    list<Statement> stmt_lst1;
    list<Statement> stmt_lst2;
    list<Mat_Comment> m_cmd_lst;
    list<Mat_Comment> m_cmd_lst2;
  end TRY_CATCH_COMMAND;

  record UNWIND_PROTECCOMMAND
    list<Statement> stmt_lst1;
    list<Statement> stmt_lst2;
    list<Mat_Comment> m_cmd_lst;
  end UNWIND_PROTECCOMMAND;

  record DECL_COMMAND
    Ident identifier;
    list<Decl_Elt> decl_elt;
  end DECL_COMMAND;

  record BREAK_COMMAND
  end BREAK_COMMAND;

  record CONTINUE_COMMAND
  end CONTINUE_COMMAND;

  record RETURN_COMMAND
  end RETURN_COMMAND;

  record SWITCH_COMMAND
    Expression exp;
    Separator sep;
    tuple<list<Switch_Case>,
Option<Switch_Case>> swcse_lst;
Option<Mat_Comment> m_cmd_lst;
  end SWITCH_COMMAND;

  record WHILE_COMMAND
    Expression exp;
Option<Separator> sep;
list<Statement> stmt_lst;
Option<Mat_Comment> m_cmd_lst;
  end WHILE_COMMAND;
end AbsynMat;
```

# Setting up a framework for model predictive control with moving horizon state estimation using JModelica

Mats Vande Cavey<sup>a</sup>, Roel De Coninck<sup>a,b</sup>, Lieve Helsen<sup>a</sup>

<sup>a</sup>KU Leuven, Department of Mechanical Engineering, 3001 Leuven, Belgium,

<sup>b</sup>3E nv., 1000 Brussels, Belgium,

mats.vandecavey@mech.kuleuven.be, roel.deconinck@3e.eu, lieve.helsen@mech.kuleuven.be

## Abstract

A model predictive control framework for optimal heating of a residential building is proposed. The control inputs are applied to a virtual building emulator model using a limited amount of measurements. State estimation is implemented using moving horizon estimation to reinitialize the states of the controller model in every time step. To implement the moving horizon estimation, the Modelica equations had to be modified. A stochastic input is declared at the controller model state equations to represent the process noise (model error). The state estimation significantly improves the output matching between emulator and controller model. The JModelica optimization framework proves to be satisfactory for this first, limited case investigated here. Future work will focus on the extension to different models and prediction errors within the framework developed here.

*Keywords: Model Predictive Control; Moving Horizon Estimation; State estimation; JModelica; Modelica*

## 1 Introduction

Building heating systems are usually controlled using a heating curve that determines the supply water temperature based on the outside temperature. An increased interest in optimal control is encouraged by the widespread adoption of optimal control in other engineering domains. Model predictive control (MPC) is a general purpose control scheme that involves repeatedly solving a constrained optimization problem. Optimal control inputs are computed using a reduced order controller model and are applied to an emulator model or a real case. Measurements of (one or more) states in the emulator model or real case are used to reinitialize all states of the controller model.

This paper proposes a general framework for MPC with state estimation using Modelica models

and JModelica. The main focus lies on the implementation of moving horizon state estimation.

In the literature, often this step is either bypassed by using the controller model as an emulator model or by assuming all states can be measured and thus the controller model is updated perfectly. However, these assumptions do not hold when using MPC in real buildings where state estimation is thus needed to update all controller model states.

Because deterministic models cannot explain the differences between the system model output and the real system observations, stochastic models are needed. Therefore, the deterministic model equations are extended with a noise term, to overcome the simplifications in the model and the input uncertainty. State estimation computes this noise term based on statistical knowledge of this extra term and system observation.

For state estimation, often a Kalman filter is considered, which updates the states, by calculating a deterministic estimate, based on the covariance matrices of the noise. A classical Kalman filter can update the states of a linear time varying model. An extended Kalman filter can update the states of a non-linear time varying model, by linearizing the model equations in the working point. The framework described in the current paper uses Moving Horizon Estimation (MHE). MHE solves a least squares estimation that determines the optimal state estimates, based on covariance matrices of the noise.

The reason for choosing MHE over Kalman filter is twofold: first, the state estimation fits in a framework that is being developed for MPC with Modelica models to be used in real buildings. The MPC framework starts with parameter estimation of greybox controller models in Modelica using the greybox building models in a model library (*FastBuildings*) developed by De Coninck et al. [1]. MHE uses the Modelica-model differential algebraic equations (DAE) formulation directly and thus keeps the great flexibility of the greybox parameter estimation toolbox. In contrast, Kalman filter is applied to ordi-

nary differential equation (ODE) models. Second, the parameter estimation, the optimal control problem and the state estimation problem are all optimal control problems which are solved using JModelica [2]. JModelica allows solving non-linear problems using gradient-based optimization. The state estimation and optimal control problem encountered here, will always be initiated with an initial guess, based on a nearly equal former solution. This will ensure that the non-linear problem will be handled robustly. This is in contrast with often reported failing of an extended Kalman filter (EKF) when handling non-linear models [3].

In this study, the MPC is applied to a virtual, single zone residential building equipped with a floor heating system and fed by a heat pump. It is represented by a ‘detailed’ Modelica model, later called the emulator model which is presented in [4][4]. Only a limited number of states are measured. The next part of the paper explains the optimization framework step by step, after which the effects of state estimation are discussed.

## 2 Optimization framework

Figure 1 schematically shows one loop MPC, which is processed every ‘open loop’ time step. The disturbance inputs are ambient temperature, global horizontal solar irradiance and internal gains. The control input is the floor heating heatflux. Since a heatflux is not a physical decision variable, it is translated into a floor heating water supply temperature setpoint as a function of the measured return temperature. The temperature of the building zone is also measured, for feedback in the high level control.

The MHE problem looks at the past time window to estimate every new initial state variables of the controller model. The MHE optimization is initialized by a controller model simulation over that past period using observed control inputs. This initialization is an important part of solving the state estimation problem, especially when the model has nonlinearities.

The optimal control problem(OCP) problem looks at the future time window to optimize the control inputs over the prediction horizon The OCP optimization. is initialized using the solution of the OCP from the past MPC-step. The shorter the open loop horizon, the closer the optimal control inputs will be to the initial values. Optimization using linear models will always find the global optimum for the corresponding linear system.

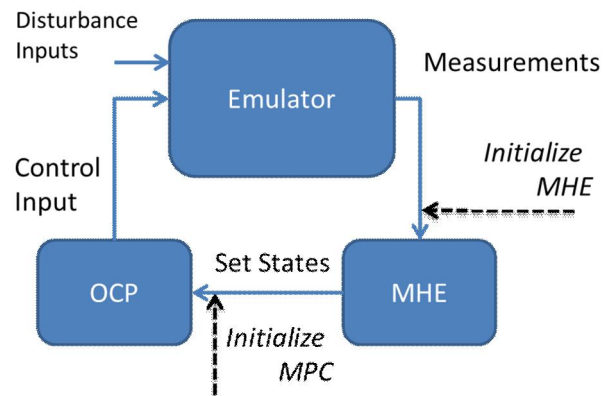


Figure 1: Outline of MPC

### 2.1 State estimation

Because of model mismatch and disturbance prediction errors, the controller model state values deviate from the emulator values. To prevent this, a moving horizon estimator is implemented to correct the states of the controller model based on measurements (or emulator model values). MHE can be seen as the dual of the MPC as it solves an optimal control problem over the past horizon to fit some measurement(s). The difference is that MHE takes the fixed past control inputs and optimizes the model error: it determines the process noise  $w$  over the optimization horizon for every state. To understand the concept of process noise, one must look at the model as a stochastic model. Take a look at the notation of a simple, explicit form of the model equations, with  $x$  the model states and  $y$  the output.  $w$  represents the model error (process disturbance/noise) and  $v$  the output error (measurement disturbance/noise):

$$\begin{aligned}\dot{x} &= f(x, u) + w \\ y &= g(x, u) + v\end{aligned}$$

The explicit formulation of the model error  $w$  means that a stochastic model allows for a difference between the state change according to a deterministic model:  $f(x, u)$ , and the state change in the real system:  $\dot{x}$ . Solving a state estimation problem means finding the difference  $w$  over the past time, such that the stochastic model accurately represents the real system behavior.

To solve the state estimation as an optimization problem, the following least squares problem should be solved by determining  $w_k$ .

$$\min_{x_0, \{w_k\}_{k=0}^{T-1}} \sum_{k=0}^{T-1} \|v_k\|_{R^{-1}}^2 + \|w_k\|_{Q^{-1}}^2 + \|x_0 - \bar{x}_0\|_{P_0^{-1}}^2$$

This is known as the ‘full information problem’, which minimizes the weighted sum of the output



noise  $v$  (on some measured outputs  $y$ ), the model noise  $w$  and the initial state  $x_0$  over the past time. The weights can be determined using the inverse covariance matrices of the model error ( $Q^{-1}$ ), the measurement noise ( $R^{-1}$ ) and the initial condition ( $P^{-1}$ ). A larger covariance of a model state thus leads to a smaller weight in the objective function and a larger value for  $w_k$  as optimal solution. A model state with large covariance will have larger deviations from the real system, which is to be expected from a larger covariance.

As time progresses, this full information problem becomes computationally infeasible. Moving horizon estimation removes this difficulty by considering only the most recent  $N$  measurements. An arrival cost is formulated to represent the information about the initial states and the measurements prior to  $N$ . The arrival cost cannot be calculated exactly (full information problem) and therefore approximations are used. For stability reasons often the prior measurements are disregarded. The initial value  $x_0$  is then set according to the solution of the last MHE update and the arrival cost is thus a constant value. Another approach is to calculate deterministic updates of the initial MHE states ( $k=T-N$ ) using an extended Kalman filter (EKF). The first approach is followed here and prior measurements are disregarded. The MHE problem translates to:

$$\min_{x_0, \{w_k\}_{k=T-N}^{T-1}} \sum_{k=T-N}^{T-1} \|v_k\|_{R^{-1}}^2 + \|w_k\|_{Q^{-1}}^2$$

or

$$\min_{\{w_k\}_{k=T-N}^{T-1}} \sum_{k=T-N}^{T-1} \|y_{meas_k} - g(x_k, u_k)\|_{R^{-1}}^2 + \|x_{k+1} - f(x_k, u_k)\|_{Q^{-1}}^2$$

With  $\|\cdot\|_A$  the 2-norm with weights  $A$ . As explained before, the weights are determined by the covariance matrices. The covariance matrix ( $Q$ ) for the is a diagonal matrix with covariance of each model state on its diagonal. The covariance matrices can be considered a tuning parameter. This is shown Figure 2 where the  $R^{-1}$  values (*weight\_meas* in the figure) takes three different values while keeping the  $Q^{-1}$  value constant. In order to get the stochastic model output to agree more with the measurements: increase  $R^{-1}$ /decrease  $Q^{-1}$ . And vice versa to agree more with the deterministic model state: decrease  $R^{-1}$ /increase  $Q^{-1}$ ). In this work, the weights are not related to the covariance matrices yet. However as we estimate the greybox models using a parameter estimation, it is mathematically possible to estimate the covariance matrices along with the model parameters [5].

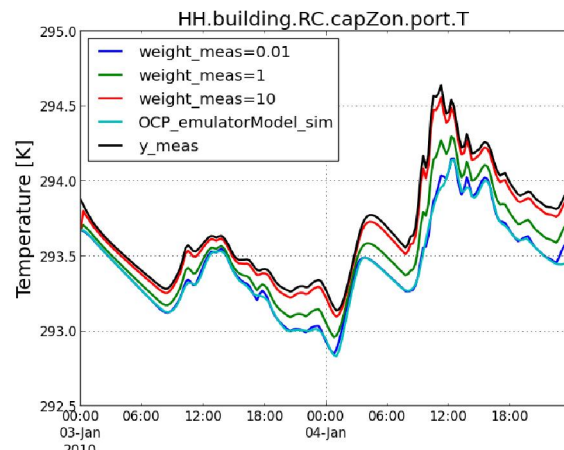


Figure 2: State estimation influence of the different weighting factors

To implement the MHE optimization problem using Modelica models, the model equations need to be adapted. Modelica model equations are by design deterministic and not stochastic. The DAE system of equations in the general implicit form is:

$$F(\dot{x}, x, u, d, p)$$

In this equation  $d$  represents the disturbance input and  $p$  the model parameters. Each equation holds and simulating a Modelica model generates a deterministic solution for the variables. To use MHE with Modelica models, we decide to add a normalized, stochastic variable  $w$  to every state equation (here for a heat capacity, with  $C=m*c$ ). The stochastic Modelica model then looks like:

```
model Capacitor_stochastic_der_paper
  "Lumped thermal capacity, with stochastic model error"
  parameter SI.HeatCapacity C "Heat capacity of element";
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port;
  input Real w;
  Real delta_T(start=0, fixed=true);
equation
  der(delta_T) = w;
  C*der(port.T) = port.Q_flow + C*w;
end Capacitor_stochastic_der_paper;
```

Figure 3: Stochastic Modelica model heat capacity

The process noise  $w$  is modelled as an input to this capacity model. The inputs 'w' of all states are optimization variables in the MHE problem. The equation shows that  $C*w$  physically represents an extra heat flux, not included in the original deterministic equation. It can also be seen that  $delta_T$  is a temperature deviation, which is the model error for the temperature (state). Since the problem is continuous in Modelica, the variable  $w$  can be determined in the MHE over the past  $N$  steps, to find the deviation  $delta_T$  of the state  $T$ . The Modelica model can be reconverted into a deterministic model for simulation purposes by setting the process noise to zero.

### 2.2 Optimal control problem

The optimal control problem is solved using a reduced order controller model. This controller model is identified based on monitoring data by use of a grey-box modeling approach. This approach starts from the Modelica library FastBuildings which defines potential low-order model candidates. For each potential model, parameter estimation is carried out, and the resulting models are compared using cross-validation, confidence intervals and other residual analyses. The best model is selected as controller model. This grey-box modeling approach is described in more detail in [1][6]. For the case investigated in this paper the time series consists of simulation data of the building zone temperature obtained by the emulator model, but this could as well be measured data. A third order resistance-capacitance (RC) model for the building zone fed by a heat pump with constant COP of 3.58 gives a good fit (rmse of 0.08 °C) to the time series generated by the emulator model. Figure 5 shows a visual representation of the

selected RC model from the FastBuildings library. There are three heat capacitors which all have a different temperature (state): capEmb.T (heater), capWal.T (wall) en capZon.T (zone). The last state is an output as it is measured in the emulator model.

$$\min_{Q_{hea}} \alpha * J_s + J_d$$

$$J_s = \int_T^T \frac{Q_{hea}}{COP}$$

$$J_d = \int_0^T (\beta * \epsilon_h^2 + \epsilon_c^2), \quad \text{with } \begin{cases} \epsilon_h \geq T_z - T_{comf,max} \\ \epsilon_c \leq T_z - T_{comf,min} \end{cases}$$

s. t.

$$F(\dot{x}, x, u, d, p) = 0$$

$$Q_{hea} \leq Q_{HP,max}$$

$$Q_{hea} \geq 0$$

$$T_z(T) \geq T_z(0) - 10$$

$$T_z(T) \leq T_z(0) + 10$$

Figure 4: multi-objective optimal control problem formulation.

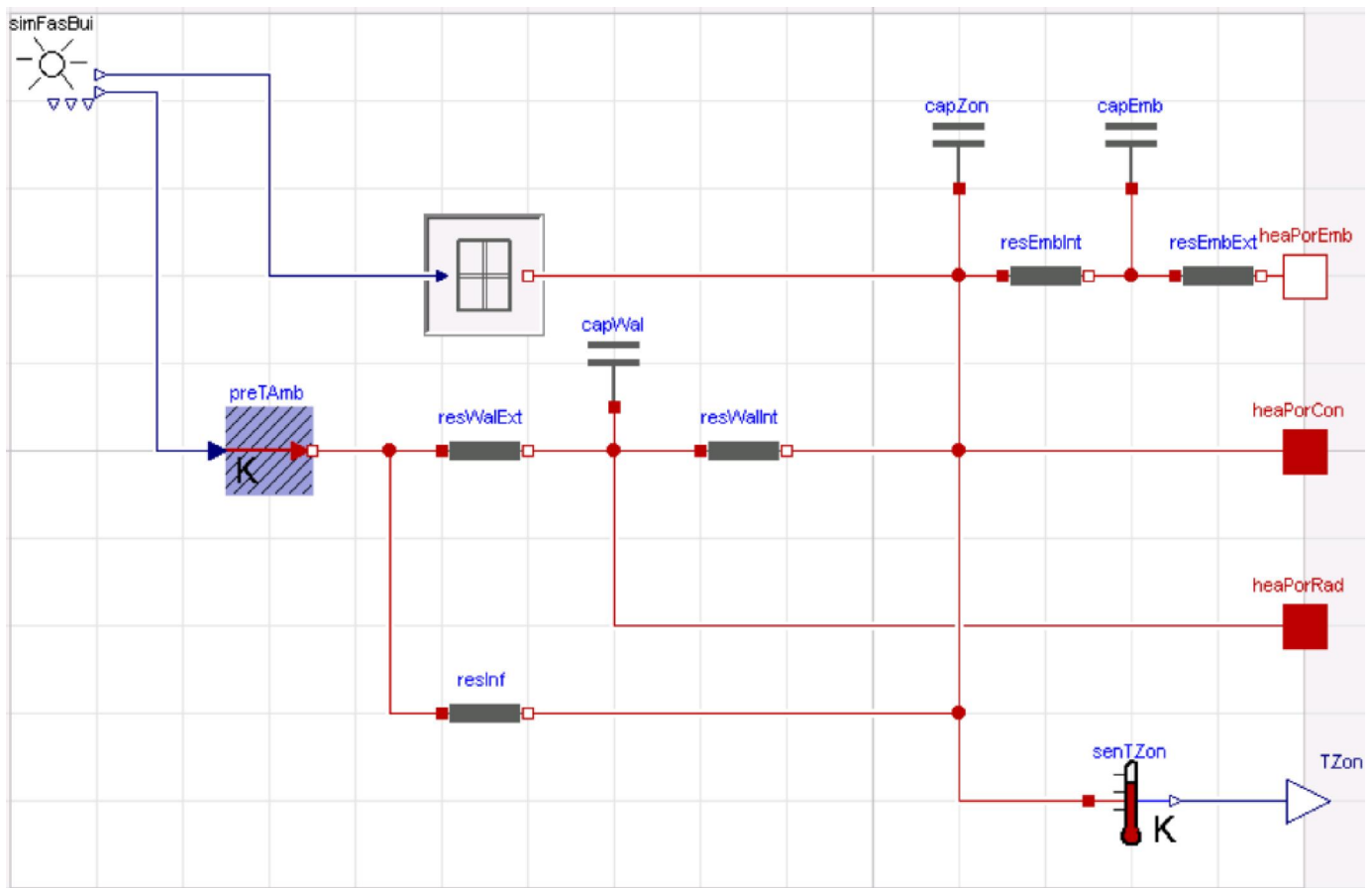


Figure 5: Visual representation of the third order greybox model in Modelica.

The optimal control problem is multi-objective. It minimizes both the energy use and the thermal discomfort and is based on a PhD regarding MPC in buildings [7]. The problem formulation is shown in Figure 4. The control input  $u$  (here:  $u=Q_{\text{hea}}$ ) determines the heat from the heating system to the zone to control the zone temperature  $T_z$ . The first term in the objective function  $J_e$  is the total electricity used by the heat pump, while the second term in the objective function  $J_d$  is the weighted sum of thermal discomfort (overheating and undercooling).

This multi-objective approach treats the discomfort boundaries  $T_{\text{comf},\text{min}/\text{max}}$  as soft, asymmetric (for  $\beta$  different from 1) constraints through the slack variables  $\epsilon$ . The external inputs (/disturbances)  $d$  are ambient temperature, solar irradiance, internal gains and the electricity price. The latter is kept constant. For the disturbances, perfect predictions are used. The optimal control input is applied to the emulator model and the building zone temperature is measured.

### 2.3 MPC

The MPC framework is written in Python, because JModelica is interfaced in Python. It is tested on an emulator model of a single zone residential building with floor heating emission system and a heat pump for heat production [4].

The future horizon over which the optimal control problem is repeatedly solved and thus over which it needs future predictions is chosen to be 2 days. It is called the ‘prediction horizon’. The past horizon over which the state estimation is repeatedly solved and thus over which it needs past measurements is chosen to be 2 days. It is called the ‘state estimation horizon’.

The future horizon over which the optimally determined control inputs are repeatedly applied to the emulator model can vary. It is called the ‘open loop horizon’. Choosing a shorter open loop horizon might improve the control, however it increases the number of optimization problems. In the example shown in Figure 6 it is chosen to be 1 day. This is a long period for control, but it is reasonable as we use perfect predictions. The figure also illustrates the horizons and the working principle by showing the zone temperature state.

First the state estimation (MHE) problem is solved to determine the initial state for the optimal control problem (OCP). The optimal control input determined in the OCP is then fed to the emulator model by translating the heat into a supply water temperature setpoint. The emulator is simulated with this heat input and the temperature of the building zone is measured. The measurement is the input to a new MHE problem.

This is shown in Figure 7, at March 2, 00:00h the states are updated. The temperatures in the controller model are discontinuously changed at every new iteration. The figure also illustrates the horizons and the working principle by showing the zone temperature state.

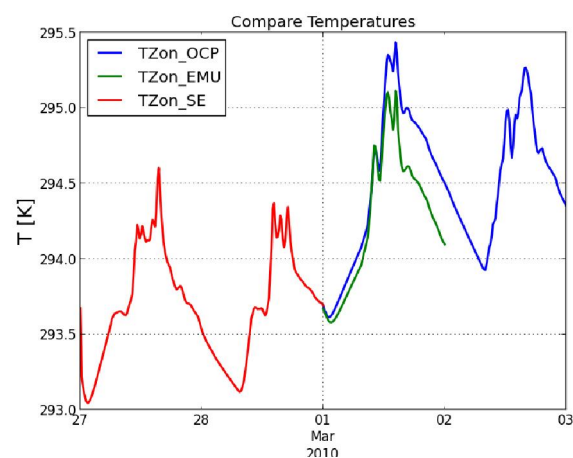


Figure 6: MPC framework, result for zone temperatures for 1 MPC iteration.

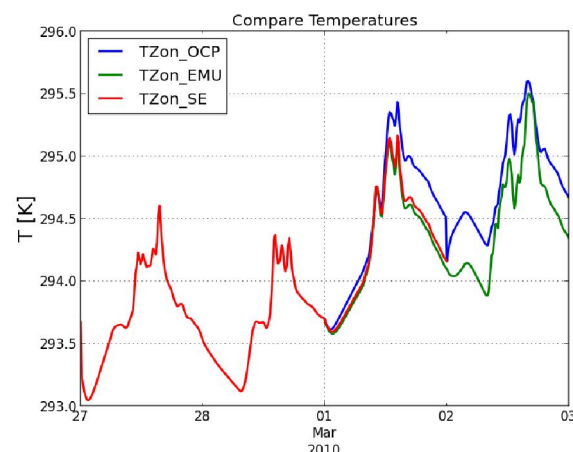


Figure 7: MPC framework, result for zone temperatures for 2 MPC iterations.

### 3 Results

The MPC is tested over a period of 10 days, with an open loop length of 6 hours. This means there will be four state updates throughout a day. As explained using Figure 6 and Figure 7, the initial values of the controller model states  $x_0$  for the MPC are determined by a first solution of the MHE. The initial values of the controller model states  $x_0$  for the MHE can be estimated freely to best capability. In order to evaluate the performance, some variations were made in the MPC formulation and settings: with and without state estimation, larger open loop (control) horizon, variation of the weighting in MHE formulation.

First a case without state estimation is solved. From is clear that the controller model would benefit from measurement feedback. The controller model (OCP) predicts a too high temperature for the states. It stays near its optimal temperature profile of around 295 K (21°C) while the emulator model (EMU) remains at a lower temperature. The third order RC model is not capable of modelling the steady state heat loss from the building zone to the soil. This third order RC model seems not appropriate for model predictive control without state estimation. The situation improves with state updates as can be seen in Figure 9. The controller model is still at its 295K (21°C), but due to the feedback of the state estimations, there is a better coupling with the emulator model. The red curve, which is in the middle of the two, represents the MHE-model state of the zone temperature. It is very close to the the measurements as the weights are chosen to be  $R^{-1} = \{10\}$ , and  $Q^{-1} = I^{3 \times 3}$ , the identity matrix. This means we assume a higher covariance on the model states than on the measurements ( $R < Q$ ). Remember that the inverse of the covariance determines the weighting and not the covariance matrix itself. If we change the values of these covariance matrices, we get a different result, which is visible in Figure 10.

The corrections made by the state estimator are visible in the optimal control problem. At every new control time step, the optimal control problem is solved again starting from the new (in this case lower) initial conditions. The lower temperature is immediately compensated by a control input determined heatflux from the heat pump to the building to regain thermal comfort.

The temperature change in the emulator does not quite follow the OCP. Firstly, this is because of a steady state heat loss to the soil and secondly because of the slow thermal response of the floor heating. A control input of 3kW does not spread as fast to the building zone as it would in the heater model.

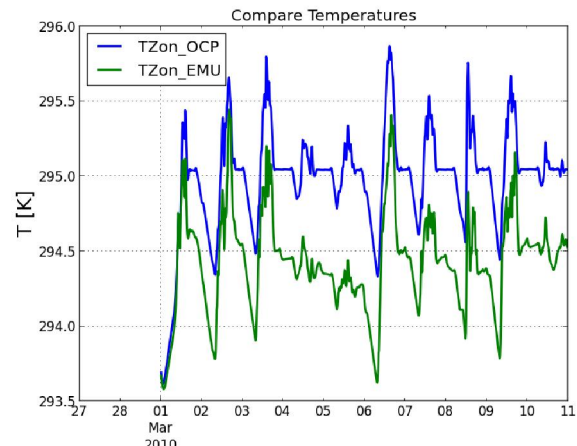


Figure 8: MPC over 10 days with no state updates.

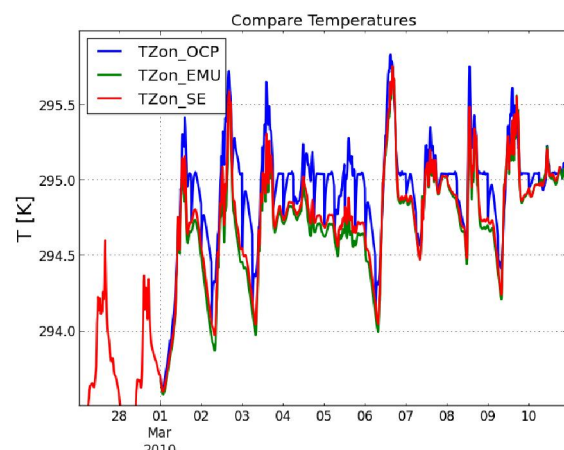


Figure 9: MPC over 10 days with 6 hourly state updates

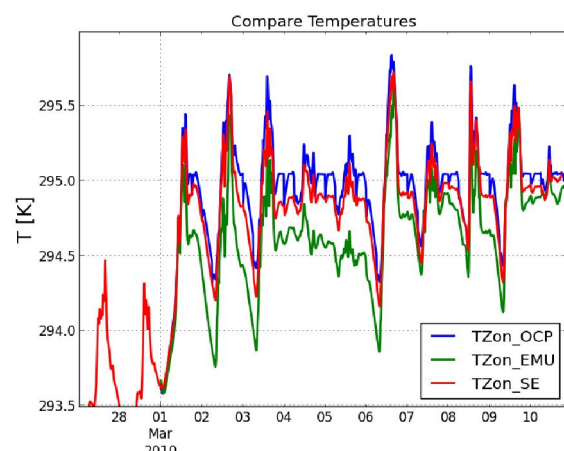


Figure 10: MPC over 10 days with 6 hourly state updates and a higher covariance for the measurements (less trustworthy)

These are two important controller model deficiencies. The first one is not so much a problem, as it can be overcome by the use of a state estimator. The second one can be misinterpreted by the state estimator as the heat to the emulator lags behind. It is therefore important to take into account the dynamics of the thermal system. If the controller time step is too small, the system might overheat as a reaction to the zone temperature measurements.

A comparison is made to MPC with a larger control time step of 1 hour update. As expected, an improved coupling pulls the emulator model building zone temperature towards the optimal values from the OCP.

In Figure 11 the coupling to the emulator model is not as would be expected. The building zone temperature of controller model is hourly updated to a higher value, and the emulator model does not seem able to catch up. One cause to this problem is found by examining the translation of the optimal control input to a physical temperature setpoint.

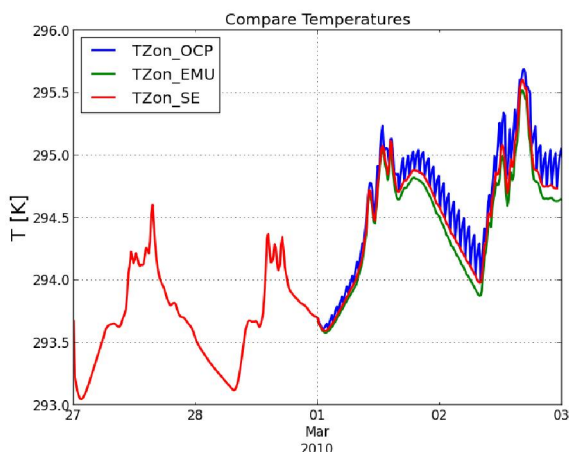


Figure 11: MPC over 1 day with 1 hourly state updates

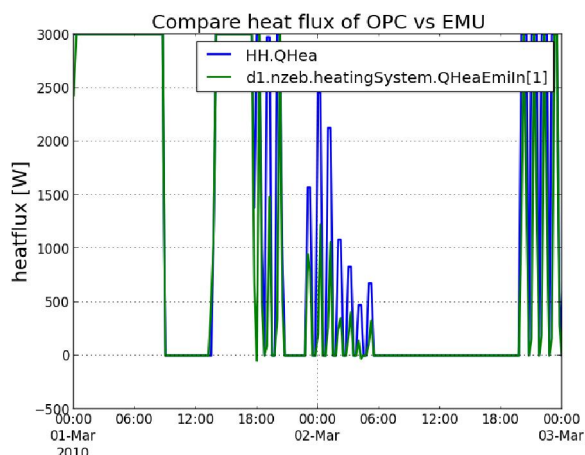


Figure 12: Optimal (HH.QHea) vs real (d1.nzeb...) heat flux

Since the floor heating system supply temperature setpoint and the heat pump setpoint are calculated using a measurement of the outlet temperature of the floor heating, a problem always arises at startup. As the outlet temperature is low when the heating starts, the first heatflux through the floor heating is always lower than the optimal control heat flux from the OCP. This effect can be seen in Figure 12. This stresses the importance of good low level control. A PID controller could help overcome this discrepancy.

To conclude we study the result of the MHE optimization for temperature difference for the states. These temperature differences are the errors of the controller model compared to the observations. In parameter estimation processes, the model errors are studied to decide whether the estimated model is 'good enough'. The decision criterium identifying a good model is whether the model error (or process noise) is white noise. The process noise produced by state estimation represents the same error of the model. This means that the controller models accuracy could be analyzed by looking at the error. The noise for the last MHE optimization problem is shown in Figure 13 for the three states.

The error is not white noise. This means there are phenomena which are not modelled. From the figure, it can be seen that the heater temperature has a steady state error with a periodic variation. This error might arise from a heat loss to the ground, which is not modelled in the controller model. The third order RC model cannot represent this loss through a model. A different RC model should thus be selected for parameter estimation. This would render a new better controller model, which will improved predictions allow better use of the optimal control horizon to improve energy savings.

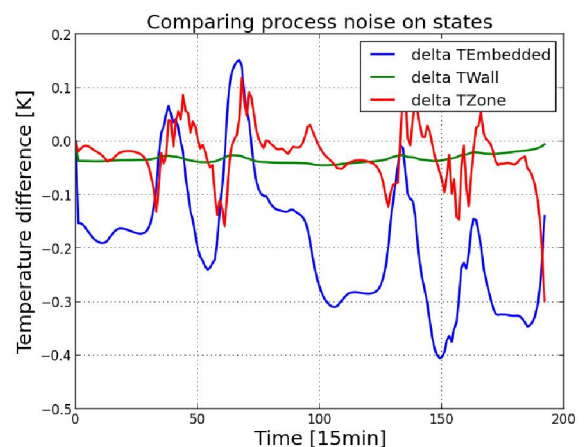
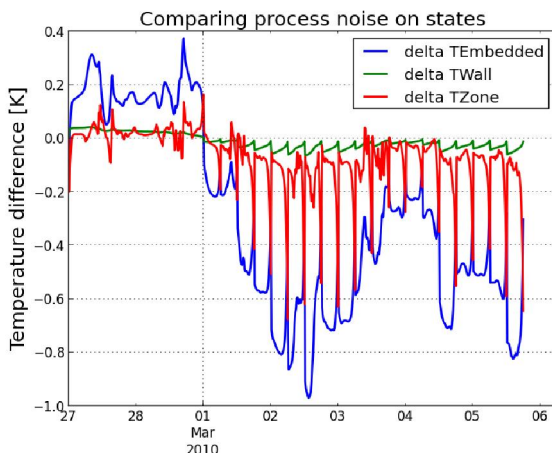


Figure 13: Process noise on states

It is also worth noting that the model error (temperature difference) in Figure 13 changes rapidly near the end. This is the case for most of the state updates of the MPC analyses with state estimation that were studied in this work. The process error on *capEmb* (state) is always high, but decreases rapidly in the end. The zone *capZon* state undergoes the inverse transition. This can be seen on Figure 14, where this last part of the results of repeated MHE solutions are shown.



**Figure 14: Process noise of the last steps in repeated solutions of the MHE problem.**

This means that in the MHE model a high noise-heatflux is present towards the embedded state (*capEmb.T*) and a high negative noise-heatflux is present to the zone state (*TZon*) near the end of the interval. This seems to mean that the temperature of the Zone is kept high until the end, and is then lowered due to a non-physical heatflux. The question rises whether this behavior is to be avoided, although it seems to be mathematically correct. Since altering the weights  $R^{-1}$  and  $Q^{-1}$  changes the behavior of the process noise for the states (and thus the timeseries in Figure 13 and Figure 14), the adoption of covariances for every states might mitigate this problem. Another solution might be to better initiate the MHE problem. This can be done by not disregarding the arrival cost, as is done now.

## 4 Conclusions

In this work, it is shown that stochastic models can be implemented in Modelica. This is an important step for optimal control framework using MPC. The MPC framework uses greybox models, which will produce an output which is different from the real building. Because not all the states in the controller model are measured states, the moving horizon estimator estimates the new initial states of the controller model. This feedback is shown in the fact that the measured state in the emulator model will be closer to the output state of the controller model. The feedback of the measurements in the emulator can be tuned by changing the weights in the objective function of the state estimator. These weights are mathematically the covariance matrices of the model states and the measurements, but since they are not always known, they can be fine-tuned to give state estimation results.

The results from the state estimation can be used to detect modeling errors or deficiencies in the controller model in the same way the residuals are checked in a parameter estimation problem. If the residuals are white noise, the controller model accurately models the system. In case the residuals are not white noise the controller model does not model the system very accurately. The state estimator can prevent the controller model from diverging from the real situation (observations).

## 5 Acknowledgements

Mats Vande Cavey wishes to acknowledge the EU FP7 project PerformancePlus (contract nb. 308991) for supporting his work on behalf of KU Leuven.

## References

- [1] R. De Coninck, F. Magnusson, J. Åkesson, L. Helsen, Grey-box Building Models for Model Order Reduction and Control, accepted for oral presentation at 10th International Modelica Conference, Lund, Sweden, 2014.
- [2] J. Andersson, J. Åkesson, F. Casella, and M. Diehl, "Integration of CasADi and JModelica.org," in 8th International Modelica Conference, Dresden, Germany 2011.
- [3] E. Haseltine, J. Rawlings, Critical Evaluation of Extended Kalman Filtering and Moving-Horizon Estimation, *Industrial & Engineering Chemistry Research* 44 (8), pp. 2451-2460, 2005.
- [4] R. De Coninck, R. Baetens, D. Saelens, A. Woyte, L. Helsen, a Rule-based demand-side management of domestic hot water production with heat pumps in zero energy neighbourhoods, *Journal of Building Performance Simulation*, 2013.
- [5] Niels Rode Kristensen, Henrik Madsen, Sten Bay Jørgensen, Parameter estimation in stochastic grey-box models, *Automatica*, Volume 40, Issue 2, pp. 225-237, February 2004.
- [6] R. De Coninck, L. Helsen, A tool chain for model based predictive control of buildings based on grey-box models, *Intelligent Building Operations* location:Boulder, Colorado, USA, 2013.
- [7] C. Verelst, Model Predictive Control of Ground Coupled Heat Pump Systems in Office Buildings, Leuven, Belgium, Ph.D. dissertation, Department of Mechanical Engineering, 2012, p 73.





## Exhibitors

### **BAUSCH-GALL GmbH**

BAUSCH-GALL GmbH (LLC) is an engineering company based in Munich, Germany, which sells and supports Modelica Libraries, works on simulation projects, organizes training courses and does consulting based on specific technical know-how. BAUSCH-GALL GmbH also offers special design services, devices and products for radio frequency (RF) applications. Based on a broad range of expertise in the solution of practical problems by effective computer application, BAUSCH-GALL GmbH serves the market for simulation and computer-aided engineering.

### **cenit**

*DRIVEN BY YOUR VISION.*

CENIT has been successfully active for more than 20 years as a leading consulting and software specialist for optimizing business processes in product lifecycle management (PLM), enterprise information management (EIM), business optimization & analytics (BOA) and application management services (AMS). The enterprise focuses chiefly on proprietary software development and on marketing standard solutions by market leaders such as Dassault Systèmes, SAP and IBM. CENIT employs about 700 staff world-wide, serving customers from the automotive, aerospace, mechanical engineering, tool and mold construction, financial services, commercial and consumer goods industries.



Claytex is an engineering consultancy and software distributor that specialises in Systems Engineering. Our expertise is in the modelling and simulation of complex multi-domain systems using Dymola and Modelica. We are based in Leamington Spa (UK) and work with Modelica and FMI on a wide variety of projects. Most recently these include the modelling of Low Carbon Vehicles, Formula 1 and Nascar Sprint Cup racing cars. These projects apply the models in a wide range of tasks including energy usage calculations, control system development, powertrain design and driving simulators. We develop a number of application libraries for Dymola include the Engines, Powertrain Dynamics, SystemID, FlexBody, VDLMotorsports and XMLReader libraries.



Concurrent Real-Time is the industry's foremost provider of high-performance real-time computer systems and software solutions. With nearly 50 years of experience in the real-time market, we deliver hard real-time performance in support of the world's most sophisticated hardware-in-the-loop and man-in-the-loop simulation, data acquisition and process control applications. With a reputation for reliability and performance, our optimized hardware and software products ensure the success of commercial and government programs worldwide. Products include the RedHawk Linux real-time operating system with guaranteed response; NightStar tools for advanced Linux debugging and analysis; iHawk real-time multiprocessors; and Simulation Workbench modeling environment. Simulation Workbench is currently used by major corporations in a variety of simulation applications including automotive component testing, vehicle driving systems, engine test stands, aircraft subsystem testing and maritime control systems. Concurrent Real-Time is based in Pompano Beach, FL, and has offices throughout North America, Europe and Asia.



CyDesign Labs was founded in late 2011 to develop a cloud platform for model-based product engineering. At CyDesign, we believe that the cloud is the future of engineering design. Our product architecture is designed from the ground up to be deployed and operated in a cloud environment – public or private, open or classified, big or small. Our primary goal is to allow engineers to configure, model, and analyze a vast array of design alternatives and to optimize their designs based on requirements.

In October 2013, CyDesign Labs was acquired by ESI Group SA, a leader in virtual product engineering. Under ESI's leadership, CyDesign is focusing on the development and deployment of model-based product engineering solutions for the global automotive and aerospace communities. CyDesign solutions are powered by CyModelica, a proprietary high-performance Modelica compiler and solver.

CyDesign Labs is based in San Jose, CA, with a subsidiary in Coventry, UK



D2T is a worldwide supplier for powertrain development and calibration, test bed engineering and equipment. With our state of the art automation, simulation and calibration tools, developed in cooperation with major OEMs, D2T is strongly involved in hybrid and electric powertrain design and validation. D2T is a subsidiary of IFPEN.

D2T is your experienced partner for:

- Modeling, simulation and design
- Control, overall supervision and calibration
- Optimization, characterization and validation



Dassault Systèmes, the 3DEXPERIENCE Company, provides business and people with virtual universes to imagine sustainable innovations. Its world-leading solutions transform the way products are designed, produced, and supported. Dassault Systèmes' collaborative solutions foster social innovation, expanding possibilities for the virtual world to improve the real world. The group brings value to over 150,000 customers of all sizes in all industries in more than 80 countries.

Dassault Systèmes' CATIA provides a fully integrated systems modeling environment that enables systems engineers to execute and analyze system or sub-systems models, while mixing dynamic and state logic behaviors, using the open source Modelica language.





IPG Automotive GmbH is one of the world's leading suppliers of simulation solutions, test systems and engineering services. Apart from vehicle dynamics simulation the simulation tools CarMaker, TruckMaker and MotorcycleMaker assist in the development of chassis control systems, driver assistance systems, hybrid technologies as well as fuel consumption analysis.



*Supporting your vision*

In the realm of system simulation, ITI is a leading developer of innovative software solutions and offers a vast range of engineering services that help to reduce time-to-market significantly. Our interdisciplinary software application SimulationX allows for comprehensive physical modeling of complex systems. Amongst others we support our customers in virtual prototyping, result interpretation and optimization of energy-efficient design. SimulationX supports the Modelica® language with open and complete CAx interfaces. The software is applied by more than 700 well-known companies, such as Audi, BMW, Bureau Veritas, Daimler, Fraunhofer-Gesellschaft, Germanischer Lloyd, Honda, Nikon, Robert Bosch, Siemens, ThyssenKrupp und Veolia.



### A Siemens Business

Since the 2010 conference, LMS considerably increased its effort to make Imagine.Lab the best-of-breed platform for system simulation. LMS' will is to deliver a combined structured approach (C-based and Modelica-based) to best serve the engineering needs, from full system to detailed component modeling over most of the mechatronics applications. LMS continues to support the establishment of Modelica as an industrial reference through its dedicated commercial support team as well as its involvement in European research projects and its support of the FMI. LMS's position in the Model Based System Engineering software market is considerably increasing to the benefit of the industry and the recognition of Modelica.



Maplesoft, a subsidiary of Cybernet Systems Co., Ltd. in Japan, is the leading provider of high-performance software tools for engineering, science, and mathematics. Its product suite reflects the philosophy that given great tools, people can do great things. Maplesoft's core technologies include Maple, the world's most advanced symbolic computation engine, and MapleSim, a Modelica-based physical modeling and simulation tool. With MapleSim, you can leverage the growing collection of industry-tested Modelica components in your own projects. Maplesoft's customers include Ford, BMW, Bosch, Boeing, NASA, CSA, Canon, Motorola, Microsoft, Bloomberg, and DreamWorks, covering sectors such as automotive, aerospace, electronics, defense, and energy.



Modelon provides industry solutions, services and technology for analytical model-based systems engineering based on the Modelica and FMI open standards. We offer unique know-how in industrial physical modeling, simulation and optimization, and model-based control design. Our customers are found all over the world and represent a variety of application areas with some emphasis on the automotive, energy and process industries. We are proud to have some of the world's best renowned technology companies among our customers. We serve our customers from our locations in Sweden, Germany, USA, and Japan.

## OpenModelica

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC). The goal with the OpenModelica effort is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. We invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.



Schlegel Simulation GmbH is an engineering company and software distributor based in Munich, Germany. Our expertise is modeling and simulation of mechatronic systems using Dymola / Modelica and other tools. We develop simulation models, work on simulation projects, realtime and hardware-in-the-loop simulations, we develop customer specific simulators and software, and provide consultancy and training. Schlegel Simulation distributes and supports Dymola.

# WOLFRAM

## COMPUTATION MEETS KNOWLEDGE

Founded by Stephen Wolfram in 1987, Wolfram is one of the world's most respected software companies. At the center is Mathematica: the world's most powerful global computation system. In 2011, Wolfram acquired MathCore Engineering AB - a founding member of the Modelica Association and an active influence in the Modelica language design since 1997. Through this, SystemModeler was released in 2012 - the most complete physical modeling and simulation tool. Unlike other systems, SystemModeler requires no add-ons, fully supports the standard Modelica model language and is designed to connect perfectly with Mathematica for the ultimate integrated modeling, simulation, and analysis workflow.



XRG Simulation GmbH

XRG Simulation has extended expertise in thermal energy system simulations in the automotive and building services field, for the aerospace and shipping industry and for power plants. We are specialized in energy engineering and support industry and research institutions in research, development and improvement of products and projects. Our excellence is:

- Modelling and simulation of thermodynamic systems
- Mathematical optimization
- Validation of models
- Software development for optimization as well as pre- and post-processing of system simulations